BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

VIDEO CONTENT SUMMARIZATION

SUMARIZACE OBSAHU VIDEÍ

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR **Bc. ROMAN JAŠKA**
AUTOR PRÁCE

SUPERVISOR **Ing. VÍTĚZSLAV BERAN, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2018

Zadání diplomové práce/19414/2017/xjaska00

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2017/2018

Zadání diplomové práce

Řešitel: **Jaška Roman, Bc.**
Obor: Počítačová grafika a multimédia
Téma: **Sumarizace obsahu videí**
Video Content Summarization
Kategorie: Zpracování obrazu

Pokyny:

1. Seznamte se s aktuálními technikami pro video sumarizaci a synopsi. Prostudujte metody zpracování obrazu a videa s ohledem na detekci lokálních změn obsahu videa v čase (lokální příznaky, sledování příznaků v čase apod.).
2. Navrhněte potřebné metody a systém, který provede analýzu snímků videa v čase a získá informace o aktivitách ve videu. Na základě těchto informací provede výběr částí videa podle požadovaných parametrů (délka výsledného videa apod.).
3. Implementujte klíčové funkce a funkční základ systému s využitím existujících nástrojů a knihoven.
4. Naplňte systém vhodnými daty a demonstrujte funkčnost řešení. Proveďte experimenty na přesnost a efektivitu řešení. Komentujte výsledky experimentů a diskutujte možnosti dalšího vývoje.
5. Vytvořte plakát a krátké video prezentující klíčové výsledky vašeho řešení.

Literatura:

- M. Sonka, V. Hlaváč, R. Boyle. *Image Processing, Analysis, and Machine Vision, CL-Engineering*, ISBN-13: 978-0495082521, 2007.
- G. R. Bradski, A. Kaehler. *Learning OpenCV: Computer Vision with the OpenCV Library*, O'Reilly Media, Inc. 2008.
- Dále dle pokynu vedoucího.

Při obhajobě semestrální části projektu je požadováno:

- Body 1, 2, 3 a částečně bod 4.

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci dřívějších projektů (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Beran Vítězslav, Ing., Ph.D.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2017
Datum odevzdání: 23. května 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstract

The amount surveillance footage recorded each day is too large for human operators to analyze. A video summary system to process and refine this video data would prove beneficial in many instances. This work defines the problem in terms of its inputs, outputs and sub-problems, identifies suitable techniques and existing works as well as describes a design of such system. The system is implemented, and the results are examined.

Abstrakt

Bezpečnostné kamery denne vyprodukujú enormné množstvo video záznamov. Ľudská analýza daného objemu záznamov je prakticky nemožná. Sumarizačný systém by bol v mnohých prípadoch veľkým prínosom. Táto práca definuje problém video sumarizácie na základe jeho vstupov, výstupov a podproblémov. Práca zároveň identifikuje vhodné techniky a existujúce práce na túto tému, pričom taktiež predstavuje návrh vhodného riešenia. Navrhnutý systém bol implementovaný a výsledky vyhodnotené.

Keywords

Video surveillance, video summarization, video processing, computer vision, activity tracking, SIFT

Klíčová slova

Bezpečnostné kamery, sumarizácia videa, spracovanie videa, počítačové videnie, sledovanie aktivity, SIFT

Reference

JAŠKA, Roman. *Video Content Summarization.* Brno, 2018. Master's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Vítězslav Beran, Ph.D.

Video Content Summarization

Declaration

Hereby I declare that this term project was prepared as an original author's work under the supervision of Mr. Beran. All the relevant information sources, which were used during preparation of this thesis, are properly cited and included in the list of references.

.....
Roman Jaška
May 23, 2018

Contents

Chapter 1

Introduction

The world is filled with surveillance cameras and their numbers keep on steadily growing with each passing day. Constant video surveillance of public places, such as city streets, busy intersections or workspaces has become commonplace. This ever-increasing number of video cameras produces enormous amounts of footage each day, most of which to never be seen by a human being. Looking for a particular event in this amount of footage is an inherently time-consuming task, which requires a lot of manual work and attention. With the ongoing improvements in video quality, the affordability of powerful computer hardware and advancements in computer vision it is possible to automate some of the tedious tasks which come with the job of reviewing hours' worth of camera footage. This work aims to design and implement a system which would aid a human operator during such endeavuor.

1.1 The concept of video summarization

Summarization in general is the act of expressing information in a concise form. This is usually achieved by omitting content which is duplicate, redundant or carries very little information. In this way, summarization is akin to *compression*. The purpose of compression however, is to minimize the space necessary to store the same information without any apparent change to its content. This does not hold true for summarization, which usually allows for some rearrangement of the content and filtering of information deemed useful.

Applying these concepts to video data, we arrive to the following conclusion. While video compression, regardless of being lossy or lossless, aims to preserve the perceived content of a video sequence while minimizing the size of the resulting file, video summary condenses the *interesting* parts of the footage into a shorter video clip.

In practice this means, that long periods of inactivity in source videos can be fast-forwarded or even skipped. Furthermore, objects not matching any relevant spatial, visual or temporal criteria can be omitted as well, depending on system's definition of what constitutes *interesting* data. We can call these criteria *constraints*. A set of such constraints can form a *query*. An example of a user query can be the following:

„Summarize, all blue cars traveling from left to right from Monday to Friday.“

The product of a summarization conforming to this query should be a comparatively shorter video sequence, containing all occurrences of activities matching the individual constraints while simultaneously suppressesing any activities not fitting the query.

Chapter 2

Motivation

Video summarization software can be a useful tool to have in a situation where there is a need to deal with extraction of meaningful information from repetitive stationary video imagery in a time-effective manner. This holds true especially in the field of video surveillance and crime investigation.

2.1 Video surveillance

Arguably the most important area day to day life where video summarization is regularly employed is the field of video surveillance. This fact is no coincidence and stems from the inherent nature of security and traffic camera footage which generally has the following traits:

1. Recordings spanning several hours
2. Long periods of inactivity
3. Stationary imagery with minimal camera movement

While the first two items pose the main problems with manual analysis of surveillance footage, the third trait can be exploited to mitigate them. The fact that the cameras do not move allows for an efficient automatization of the analysis process. As such, the system designed and implemented as the subject of this work works with the basic assumption that the input footage is recorded by a stationary camera. Furthermore, the focus will be placed on analysis of traffic footage. Let us examine a fictional story where the usefulness such tool can be exemplified.

2.1.1 Stolen vehicle scenario

Consider the following scenario; a red van was reported stolen on Friday. The vehicle was last accounted for on Monday in a warehouse parking lot. The area where the warehouse is located has 5 exits, each of which equipped with a continuously recording surveillance camera. Now imagine that you are tasked with reviewing all the footage from said cameras. Since we have 5 cameras, recording around the clock for 5 days, you are left with the worst-case scenario of 600 hours or 25 days' worth of footage to review. You could employ a small group of people to review the fast-forwarded footage, but even this would consume a considerable amount time and resources. These people would also be prone to overlooking

the pertinent events in videos, since fast-forwarding usually works by skipping entire frames of a video, and thus possibly discards the information you are looking for. Not to mention that human operators looking at these recordings would very likely grow less attentive and thus more erroneous as time passed on.

The number of man-hours spent reviewing the footage as well as the probability of detection could be drastically improved if we introduced a video summarization system into this situation. The said system would analyze the footage, process the search query and identify all instances of red vans leaving the area and condense them in short videos corresponding to each recording location. These could be checked by one person within a few minutes. The knowledge of which exit was used to steal the vehicle would allow to narrow down the direction of further search using traffic cameras and possibly lead to quick location of said vehicle. The plausibility of the technical aspect of the solution will be the subject of following chapters.

2.2 Other applications

Video summarization has other potential applications aside from situations similar to the scenario described above. Summarized sequences are suitable candidates for long term archival of surveillance footage or other instances of static video recordings with sparse periods of activity. This way, instead of storing several hours of raw video, one could store only the short, summarized versions and reduce the storage space necessary for archival by a substantial amount. Exclusive archival of summarized videos, while great for storage space requirements, would however lead to a reduced possibility of further application of user queries on the video data. For this purpose, the preferred solution would be to store the pre-processed versions of videos. Using this approach, the amount of data stored would still be reduced when compared to raw data but the ability to further refine the summary could be preserved in its entirety. The only trade-off would be a slight increase in size caused by the need to store the detection metadata along with parts of the source video sequences. The amount of achieved savings would of course be highly dependent on the amount of activity in the source data, with less active videos providing better gains, simply because of the reduced length of a summarized video.

Another possible application is the usage of video summary as a pre-processing step for other, more demanding forms of video analysis. Running a simple movement-based summarization before application of a computationally demanding task could help speed up given process.

Chapter 3

Previous works

This work is inspired by a series of previous publications by Prof. Shmuel Peleg, et al. from *Yissum Research Development Company of the Hebrew University*. The very concept of video summarization resulting in another video, called *Video Synopsis* in the article, comes from their 2006 piece of work titled *Making a Long Video Short: Dynamic Video Synopsis*. In this paper, the researchers state the following:

> *„Video browsing and retrieval are inconvenient due to inherent spatio-temporal redundancies, where some time intervals may have no activity, or have activities that occur in a small image region. Video synopsis aims to provide a compact video representation, while preserving the essential activities of the original video. We present dynamic video synopsis, where most of the activity in the video is condensed by simultaneously showing several actions, even when they originally occurred at different times.“ [15]*

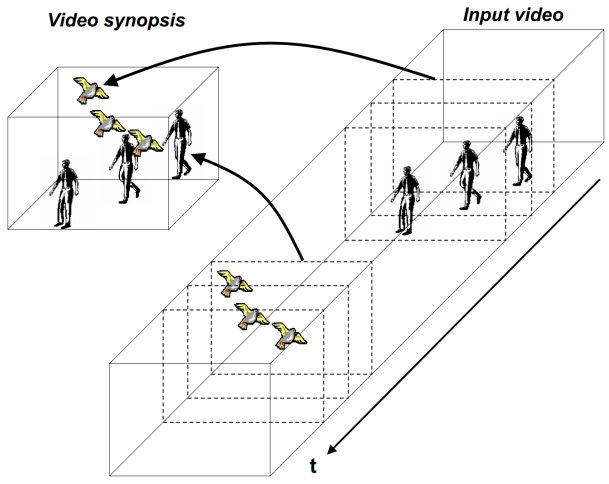

Figure 3.1: An illustration of a *video synopsis* from the original paper. [15]

In this paper, the researchers present a novel approach to video summarization. At the time of its publication, various video abstraction methods have already existed. These preexisting methods however only focused on the generation of a set of images or short clips containing interesting activity. Both approaches were key-frame based, meaning that

the basic building block of the output was an entire frame of the input video. In another approach, a static summary image was produced as a mosaic of interesting parts of the input video.

The novelty of Prof. Peleg's method was in its focus on the creation of an entirely artificial summary *video* rather than a set of images or clips.

The article presents two methods of achieving video synopsis. In the initial approach the article describes, the result was achieved using pixel-wise summarization. Input pixels were flagged as *Active* based on their temporal value changes. Subsequently, the active pixels were shifted in time using energy minimization, with a cost function crafted in such way, as to avoid discontinuities and loss of active pixels. This low-level approach produced summaries containing a stroboscopic effect, meaning that the resulting videos had the possibility of multiple instances of the same object being present on the same output frame. Furthermore, since no context information was known about objects comprised by the pixels, the content of the summarization could not be easily filtered.

The second approach described, aimed to solve this problem using a higher-level object-based summarization. In this approach, *Activity strips* are detected. These strips represent a subset of detected areas in frames with corresponding activity. The resulting summariztion is achieved by shifting the entire strips instead of individual pixels. This is the approach that will be implemented in my system. This problem is described as computationally intensive, so a number of restrictions is introduced in attempts to reduce the number of possible solutions.

In their following paper on this subject, titled *Webcam Synopsis: Peeking Around the World*, the researchers expand on their previous ideas and introduce the idea of query-based synopsis.

> *„We would like to address queries like 'I would like to watch in one minute the highlights of this camera broadcast during the past day'[13]"*

The concept is even further expanded in the 2008 paper *Nonchronological Video Synopsis and Indexing*, where the idea of breaking the chronological order of source activities is discussed.[14]

Finally, the paper *Clustered Synopsis of Surveillance Video* from 2009, written by the same authors describes the possibility of clustering of the detected activities by similarities of properties. The authors found, that this clustering allowed for more coherent results with less confusing outputs.[12]

In 2010, a company by the name of *BriefCam, Ltd.* was founded by Prof. Peleg and others. The company procured a license to use the technology developed at Yissum and currently offers a commercial solution based on this research. The *History* section of the company's website states the following:

> *„The technology was reported to have been used in the investigation of Oslo bomber and mass murderer Anders Behring Breivik, and the Boston Marathon bombing investigation."*.[18]

Chapter 4

Theoretical background

The aim of this chapter is to briefly introduce the reader to the basics of each concept necessary for understanding of upcoming chapters as well as to discuss my choices of particular techniques used in the implementation.

4.1 Background subtraction

Background subtraction is the process of extraction of foreground objects from an input image. Background subtraction of a stationary video sequence is a common first step in many computer vision tasks. It is highly suitable for movement detection. As such, a wide variety of techniques have emerged over the years of research on the subject. The general steps however remain the same for most of the approaches.

At the heart of each technique lies a model of the background. These models can range from as simple as the previous frame, an average or median of previous samples to statistical models or even neural networks. [2] This background model is usually being continuously updated as new frames are processed to account for gradual changes in appearance of the background. With each new frame processed, the subtractor performs a differencing operation, comparing its model to the real frame. A low amount of Gaussian blur is often applied on the new frame before this step, in order to cancel out the small amount of variance caused by noise. The resulting difference forms the foreground mask (Figure 4.1), which is a binary image representing the presence of an object in the foreground. Shadows can also be detected as they usually only decrease the brightness of an area but preserve its visual structure. Once shadow detection is introduced, two binary masks are necessary, one for the objects and one for the shadows, unless a grayscale representation is used instead of binary.

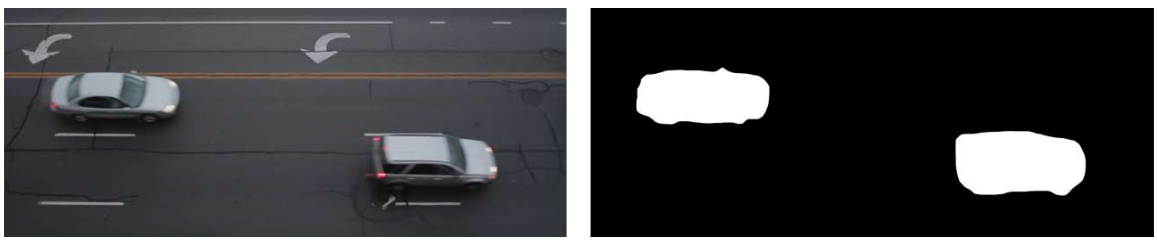

Figure 4.1: Input frame (left) and extracted foreground mask (right)

Having tested the various background subtraction methods available in the OpenCV library, I have determined that the mixture of gaussian distributions model described in the paper *An Improved Adaptive Background Mixture Model for Realtime Tracking with Shadow Detection* [4] will be the most suitable for the needs of this project.

4.2 Object tracking

Object tracking is another common computer vision task, having multiple approaches and documented techniques. The basis of an object tracker is a point tracker. In order to track an object multiple points of interest are identified and tracked as a whole. The purpose of a general object tracker is to maintain the identity and determine the position of a given object across multiple frames of a video. An object tracker needs to be initialized by a specification of the area to be tracked. Once initialized, the tracker will update its internal state with each frame, searching for the new position and analyzing the changes in appearance of the tracked object. Multiple traits of the detected objects are exploited in order to increase the precision of a tracking algorithm. These include attributes like positional history, size, speed, direction or visual features. Important features of an object tracker are:

- Ability to preserve identity of tracked object across frames
- Ability to cope with temporary occlusion of the tracked object
- Resiliency against false positives and drifting
- Reliable tracking failure detection

For this project I have decided to use the *Median flow* tracker, described in the paper *Forward-Backward Error: Automatic Detection of Tracking Failures*[5], available in the OpenCV library. This tracker has provided the best results in my initial test cases.

4.2.1 Median flow tracker

The Median flow tracker is based on evaluation of a median of the vectors generated by the changes in points tracked using the *Lucas-Kanade* method. It is a forward-backward tracker, meaning that it maintains the historical trajectory of the object for analysis. It works under the assumption that the tracking consistency should be independent of the direction of time-flow. With each new sample, the object is tracked forward in time, as well as backwards. These two trajectories are then compared, and the discrepancy is evaluated. This allows for a reliable detection of tracking failure.[5] The algorithm works best when the tracked objects move rather predictably.[20]

These attributes of the Median flow tracker make it a suitable solution for the needs of this project. This is due to the fact, that the tracked objects will be primarily vehicles, which are highly predictable and the timely detection of tracking failure will allow for early termination of tracking to avoid identity inconsistencies.

4.3 Visual features

Features in the field of computer vision can be described as the most unique parts of an image. The ability to identify, describe and compare these unique parts is the basis of a

vast number of computer vision tasks. In this section I will describe the most commonly used type of image features - the SIFT features. These are also used within this project for the purposes of identification of visually similar objects as described in the section *Bag of Words* 4.3.1.

4.3.1 SIFT

The *Scale-invariant feature transform* consists of two main parts. First is the detection of keypoints in an image. Subsequently an efficient way to describe these points is needed. The measure of efficiency in this case is primarily the robustness and speed of comparison of these descriptors. The main advantage of SIFT, when compared to earlier methodologies, is its scale invariance. This means that SIFT descriptors, detected at different scales, can be compared with fairly consistent results. SIFT was developed by David G. Lowe, and was first described in his 2004 paper titled *Distinctive Image Features from Scale-Invariant Keypoints*. The paper was a ground-breaking success and has been cited more than 45 000 times since its publication [7]. The method itself is currently patented.

Detection of keypoints

First a scale space is constructed. Scale space is a set of images produced from the input image by application of a Gaussian filter of different σ values at progressively lower resolutions. This produces octaves. Each octave's images have the same resolution, but different σ values. The paper recommends a particular combination of σ values and number of octaves.

Once the scale space is created, a Laplacian of Gaussian is needed. LoG is suitable because it performs well as a corner detector and corners in turn make for good keypoints. LoG however is a quite costly operation to compute and thus an approximation is used. This approximation is achieved by performing a simple subtraction of each neighboring pair of images in an octave. This technique is called the Difference of Gaussians, or DoG for short. Performing this operation on each octave, we get a DoG of multiple sizes.

The next step is the actual detection of keypoints. This is done by detecting the extremes in the generated DoG images. The image is analyzed pixel by pixel. Each pixel's surrounding values are then compared with its value. Pixel's surroundings are comprised of its 8 immediate neighbors as well as the 9 pixels above and below in the neighboring scales space images. This leaves us with 26 neighbors to compare. Since only extremes are of interest, most of the comparisons are skipped, because an extreme must have a value that is higher or lower than all of its neighbors. The actual extremes often occur between the pixels and thus a Taylor expansion around the point is necessary.

The found keypoints are then filtered according to their gradients in order to remove any keypoints occurring on the edges of the image and on flat regions. This reduces the number of keypoints but increases their overall stability. Since the information about the scale at which a point was detected is known, the points are scale invariant.

Descriptors

The detected keypoints need to be described in a suitable way. SIFT does this in the form of *features*. A feature, in terms of SIFT, is 128-dimensional vector. In order to obtain this vector, the following steps are taken. The area around the detected keypoint is analyzed for gradient directions and magnitudes. These are collected into a 36-value histogram with

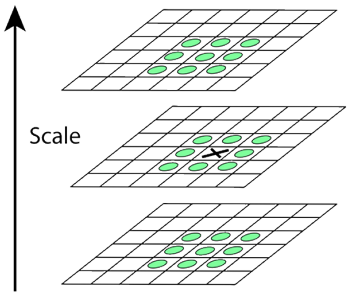

Figure 4.2: Visualization of a point neighborhood [7]

bins representing 10-degree direction steps. The magnitude of each gradient is added to the corresponding column. This step ensures rotational invariance since further steps will be computed relative to these values.

Next a 16x16 area around the keypoint is sampled. This area is further divided into smaller 4x4 regions. In each of these regions the gradient directions and magnitudes are distributed into an 8-bin histogram, with each bin corresponding to 45 degrees. Finally, a Gaussian weighting function is applied to the resulting window. These 16 groups of 8 values each form the 128-value vector, which is the SIFT feature descriptor. [7, 16]

The OpenCV library provide and implementation of SIFT.

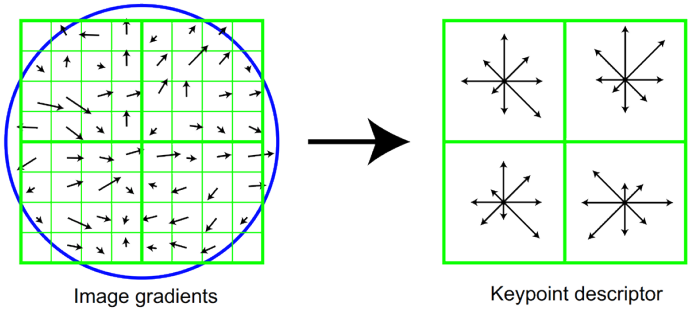

Figure 4.3: A simplification of a SIFT descriptor. The size of the window is 8x8 instead of 16x16 for clarity. The blue circle represents the Gaussian weighting function. [7]

4.4 K-means clustering

K-means clustering is an unsupervised, iterative process of sorting data points into a defined number of groups, based on their similarity. The number of final groups is defined by the constant k. The clustering process starts with specified, or randomly determined centroids of the individual clusters. Two steps take place in each iteration of the process. First, each data point is assigned to its closest cluster centroid. A regular Euclidean distance is employed for calculation of this value.

Secondly, the position of each centroid is updated, using the mean of all of its assigned data points. These two steps are executed repeatedly until a terminal condition is satisfied. Examples of this condition include the number of iterations or stabilization of the cluster

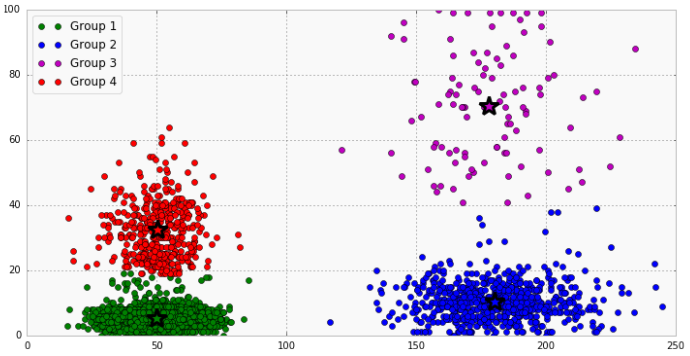

Figure 4.4: An example of the result of k-means clustering on a set of two-dimensional data points clustered into four groups. The stars represent centroids of the clusters.[17]

centroids. This method is known to converge on a result, but the result may not be optimal. [17] A change in the initial centroid positions can affect the final results. A k-means clustering solution is available in the OpenCV library.

4.5 Bag of Words model

Bag of Words is the name of a technique stemming from the field of text processing. This model was originally conceived to simplify text for classification purposes.

The first step towards implementing a BoW model is the acquisition of a vocabulary. The vocabulary is a set of words, or groups of words, that are deemed relevant in the text. Once a vocabulary is established, the frequency of each vocabulary entry encountered is counted into a histogram with bins corresponding to the vocabulary words. This results in a vector of size equal to the vocabulary size. This vector is the resulting „bag of words“, and can be compared to other such vectors, obtained using identical vocabulary.

This idea can be successfully applied to the field of computer vision. A slight variation on the name is used: *Bag of Features*, or *Bag of Visual Words*. As the names suggest, we are no longer working with literal words. These are instead replaced with suitable image features. These can range from simple samples of small areas of images, to full-blown feature descriptors like the SIFT features described in section 4.3.1.

The *visual vocabulary* is obtained by the means of clustering. A large number of features is sampled from a training set. The contents of this set will determine the accuracy of the resulting vocabulary. A number n is decided, which is used to build the vocabulary. The sampled features are then clustered into n clusters, using the k-means method described in section 4.4. These clusters represent the visual vocabulary. The number of input samples and the number of clusters directly affect the resulting accuracy and robustness of a vocabulary.

Once trained, the input images are described using the visual words of the trained vocabulary. A number k is decided, and k features are extracted from the image being described. The type of features extracted needs to match the type of features used during the training process. Once these features are extracted, is placed into the histogram bin of the closest vocabulary word. The resulting histogram of length n and sum of k is produced. This histogram represents the resulting description of an image and can be efficiently compared to other such histograms for similarity.

The method has many positive traits advantageous to this project. These are the relatively small and adjustable descriptor size, efficient comparison of described images and the ability to train a task-specific vocabulary. However, there are also downsides. The vocabulary is fixed and once it is changed, all the images described using this given vocabulary need to be described from scratch using the new vocabulary. Another difficulty may arise from the fact that the effectiveness of the comparison is largely determined by suitability of the training samples. A vocabulary trained on images of vehicles will perform inadequately when applied to images of animals.

The OpenCV library provides an implementation of a BoW model described in the paper *Visual Categorization with Bags of Keypoints* [3].

4.6 Cosine similarity

Cosine similarity is a widely used similarity function, used to determine the similarity of two, n-dimensional vectors. This similarity is determined by the computation of the cosine of the angle between these two vectors. This angle will range from 0 to 180 degrees, regardless of dimensionality of the space. The cosine distance can be calculated by computation of the cosine of the angle, followed by application of the arc-cosine, in order to translate the angle to the 0-180 range.[6]

The result is then projected onto the unit circle, resulting in value range of -1 to 1, or 0 to 1 if the direction is not of concern. The two vectors in question are deemed most similar, when they are parallel with each other, resulting in the cosine of their difference being 1. On the other hand, the vectors are considered dissimilar if they are orthogonal to each other, resulting in the cosine value of 0.

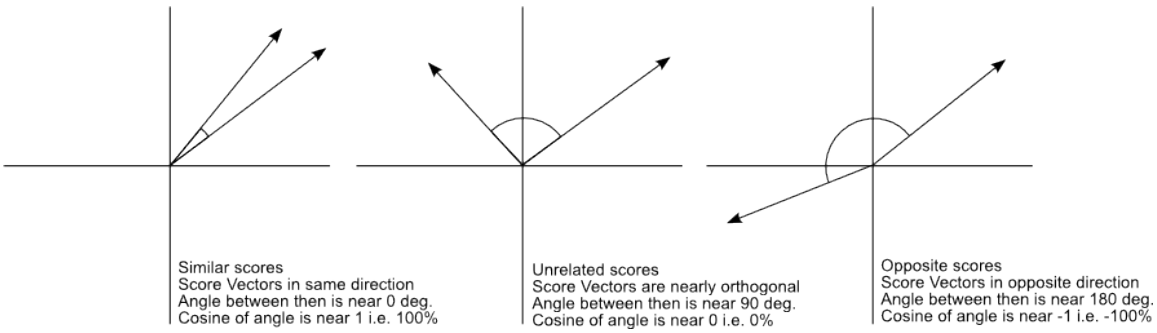

Figure 4.5: A visualization of possible outcomes of the vector comparison, demonstrated in a two-dimensional space. [11]

The advantage of this approach over Euclidean distance, is the fact that the magnitude of the vectors does not affect their similarity. The consequence of this, is that values of vastly different ranges can still be considered similar, if the direction of their vectors is in agreement.

This operation can be efficiently implemented without the need for computation of actual trigonometric functions. Instead, a vector dot product can be calculated, and divided by the L_2-norms of x and y, which in this case are the Euclidean distances from the origin. [6]

Chapter 5

Problem definition

Before we venture any further into the subject of the individual steps of video summarization, it is necessary to properly define the problems it presents and outline a high-level structural overview of the system designed to solve them. Each of these elements and their respective problems will be described in this chapter.

5.1 Problem structure

I have a large number of long videos, that I would like to be able to produce a video summary of, as described in section 1.1. In order achieve this, a video summarization system needs to be designed and implemented. This system and its subsystem will be defined in terms of their inputs, inner processes and outputs.

The system needs to perform two basic tasks. It needs to be able to process the input groups of video files in search of any activity. This is the first part, that I will refer to as the *Analysis phase* from this point on. Upon analyzing the input files, the results of the analysis need to be stored in a suitable way for later processing.

Secondly, the system will need to obtain and evaluate user queries. This will be referred to as the *Response phase*, borrowing the term from the second paper on the subject of *video synopsis* by Prof. Peleg, et al.[13] Upon evaluation of user queries, the final subset of activities obtained from the analysis phase is known, and thus a summary can be assembled.

5.2 Analysis phase

This phase represents the initial entry point of the entire summarization system. It is a self-contained phase which will only need to be executed when new video sequences enter the system. It can be fully automated.

5.2.1 Inputs

The inputs of the analysis phase are comprised of multiple groups of long video files. Each group can contain one or more video files. These files must adhere to a special set of constraints. The footage is required to be static in nature, meaning that it must be recorded or rendered, when dealing with artificial imagery, using a fixed camera.

The secondary requirement for the video sequences in an input group is a strict correspondence of individual files' technical attributes. The frame rate and resolution of the

input files must match in order to avoid additional steps during the response phase. These conditions are implicitly satisfied when dealing with a group of video files provided by a single camera.

These were the general assumptions about the input files that I will be working with throughout the rest of this project.

5.2.2 Inner processes

Each input file needs to traverse an analysis pipeline. The entire length of the video file needs to be analyzed in a time-efficient manner. Following problems need to be solved during the analysis of a single input file:

1. Object detection - Once a new object enters the frame, it needs to be reliably detected. The sensitivity of detection should be such, that no object is missed. This may lead to an increase in the number of false positives, which will need to be identified and discarded later on.

2. Object tracking - The detected objects need to preserve their identity across the entire span of frames they are present in. An efficient object tracking solution is necessary. Early termination should be prioritized over false tracking matches which may result in an unwanted merging of different objects' activities.

3. Activity description - After the tracking period concludes, the detected activity needs to be finalized. This finalization process should generate a useful description of the activity for later stages of processing, namely the response phase.

4. Description storage - Once the activity is finalized and a fitting description is produced, it needs to be stored using a sufficient a long-term storage solution.

5. Background approximation - The final step of analysis phase is the acquisition of a background image. This image represents the scene devoid of any moving objects. Since there is no guarantee, that a period of absolutely no activity will appear in the input video, the background image will have to be approximated. The background image is vital for later stages of the summarization process, where the detected activities will be projected onto it.

5.2.3 Outputs

The analysis phase results in a preprocessed representation of activities detected in the input video sequences. This representation and its storage solution should be carefully selected for space efficiency, since large quantities of activities are expected to be handled. The second consideration is the ability quickly retrieve any desired activity matching a given set of constraints and compare it to other activities.

The secondary output of the analysis comes in the form of a background image described in the previous section. The quality of this image will directly affect the perceived visual quality of the final summary, but will have no effect on the effectiveness of the temporal realignment of activities themselves. This concludes the analysis phase of the problem.

5.3 Response phase

This phase represents the point at which user interaction with the system will be required. User specified queries will be applied on a subset of the data accumulated from the analysis phase. These activities will be further filtered and eventually rendered into the final summary.

5.3.1 Inputs

The response phase picks up where the analysis left off, meaning that the produced representation of analyzed activities will be used as a primary input. However, secondary inputs will still need to be specified by the user. These will fall into two categories.

The first is the selection of analyzed videos upon which all of the subsequent operations will take place. The other will pertain to the activities themselves. This is where the various queries and views will be applied to the data. The type of these queries can range from spatial, visual to temporal.

5.3.2 Inner processes

After a number of processed videos is selected as an input for the summary, additional problems need to be solved in the following order:

1. Activity merging - Once the user defines the subset of files to appear in the summary, the activities originating in these files will need to be merged into a single pool of activities upon which the next steps will take place.

2. Constraint specification - In order to be able to filter the activities, the user will need to be provided with a way to build their queries, using individual constraints. Some of these are of visual nature and thus, a graphical user interface will be necessary.

3. Constraint application - Once defined, the user constraints will have to be aggregated into a single query that will be applied on the selected data in order to retrieve matching activities.

4. Temporal realignment of activities - After the final subset of included activities is known, a temporal shift for each activity needs to be determined in order to fit the activities together as close as possible.

5. Summary rendering - The generated offsets can now be applied to each activity. Activities will be projected onto the background obtained during the analysis phase. This produces the frames of the final video summary.

5.3.3 Outputs

The output of a video summarization process is once again a video sequence. The output video will only contain activities matching the user specified constraints. Should no constraints be specified, the output will contain all of the detected activities. The selected activities will be overlaid onto a background image in such way, that the output video will be considerably shorter than the sum of its inputs, while simultaneously displaying activities from different files and temporal positions, omitting any periods of inactivity.

Chapter 6

Proposed solution

This chapter describes the design of a system which aims to solve the problems laid out in chapter 5. This system is comprised of a set of interoperable sub-systems and components.

6.1 System structure

I have identified and designed the following parts of the system:

Video storage
: The actual location of the input data as well as the destination of rendered summaries. This may be a local filesystem directory or a remote network-attached storage. This storage will be expected to be accessible during the entire summarization process. Further details are available in section 6.2.

Input analyzer
: This part of the system closely corresponds to the *Analysis phase* described in section 5.2. Files are retrieved from video storage and processed. This part will require no user interaction apart from launching and will be implemented in the form of a class library with a companion console application. This part is described in section 6.3.

Database
: As mentioned in section 5.2.2, a long-term storage solution is required for the detected metadata. A regular relational database will be used to satisfy this requirement. The details of this choice and the design of the database itself are further expanded upon in section 6.4. This database will facilitate all communicational needs of individual parts of the entire system and serve to provide a persistency layer for the application.

Query processor
: This part of the system will be responsible for the evaluation of user queries mentioned in section 5.3.2. The description of these queries and individual steps of their evaluation are laid out in section 6.5.

Output assembler
: The output part of the system, having two main functions. First is the assembly of the final alignment of activities using temporal shifts, with the second being the actual rendering process of the output video. These processes are explained in sections 6.6.1 and 6.6.2 respectively.

GUI application

A graphical user interface will be required for the purposes of user interaction mentioned in section 5.3.2. This application is the main communicational channel between the user and the entire summarization system. The application is situated in the *Response phase* of the summarization process as described in section 5.3.2. The design of the user interface is described in section 6.7.

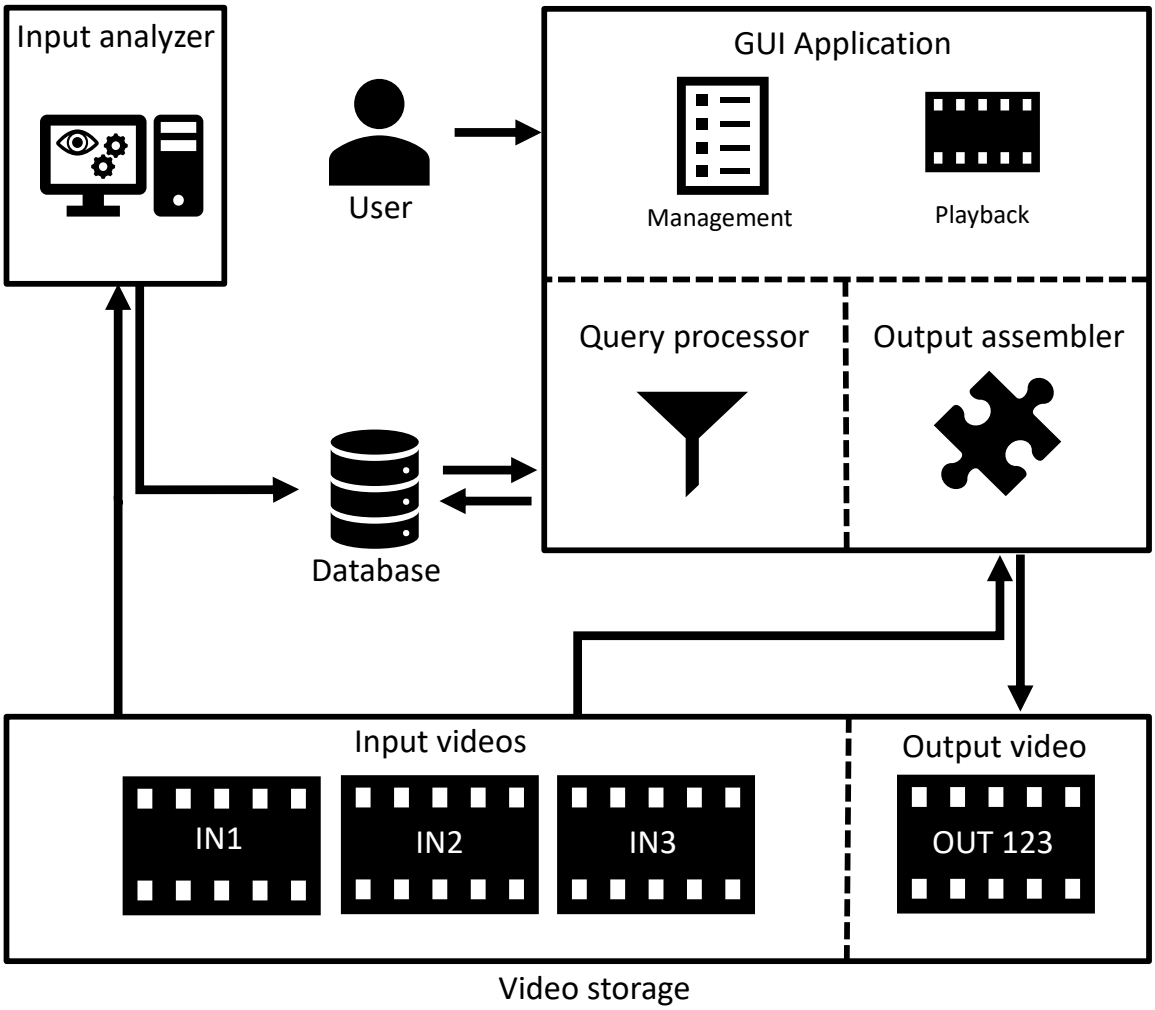

Figure 6.1: Diagram of the proposed system. Arrows represent the direction of data access and communication between individual elements.

6.2 Video storage

The source video files will need to be accessed during various stages of the summarization process in order to minimize the spatial requirements of analysis outputs. As such, almost no visual data will be kept. Instead, only metadata relative to the source files will be stored and the actual image data will be loaded from the source files as necessary. This approach

requires the source files to not change their location after they have been processed by the system, until a video summary is produced, and the files are determined to be of no interest for further processing.

The rendered summaries will be stored alongside the source files, with corresponding filenames. A suitable suffix will be added to distinguish the individual summaries. As for the actual storage solution, any regular filesystem supported by the host operating system should be sufficient for the needs of this project.

6.3 Input analyzer

The first part of the system is the input analyzer. This component takes the initial input in the form of a single or multiple recorded video files and produces processed *Tubes*.

6.3.1 Tubes and Slices

A *Tube* is a three-dimensional representation of a detected activity. The actual term *Tube* comes from the previously mentioned paper *Webcam Synopsis: Peeking Around the World* [13]. However, the basic idea behind tubes can be traced back to earlier work of the same authors, where they are called *activity strips* [15]. In order to better understand the concept of a tube, let's imagine a simple video of a car driving by.

In each frame, the car occupies a certain part of the two-dimensional frame image. As the frames advance, the car changes its appearance and position in the frame. We can call each of these occurrences of the detected object a *Slice*. Now, if we take all of these two-dimensional slices as they move across the 2D space and time, we are able to construct a three-dimensional tube. The three dimensions in this case correspond to the X and Y dimensions of the input frame, with the third dimension being a representation of time.

6.3.2 Analysis pipeline

Once the analysis of a video is commenced, the following steps are taken on a frame-by-frame basis.

The analysis takes place on two levels; detection and tracking. Since the analysis needs to be fully automatic, the tracking portion of the pipeline must to be guided by the detection portion. The right time to start and finish tracking must be determined automatically.

1. Frame resizing - In order to reduce the computational cost of frame analysis, each frame is resized to match a predetermined resolution. All other operations are executed using this resized frame.

2. Detection - The detection part of the pipeline is responsible for the extraction of individual moving elements from a frame.

   (a) Background subtraction - A background subtraction operation, as described in section 4.1, is applied on the resized frame. This operation results in a coarse binary foreground mask which requires further processing.

   (b) Morphological operations - The acquired foreground mask usually contains unwanted noise and needs to be refined using morphological operations. A morphological opening and closing are applied to the foreground mask. This rids the mask of small patches of activity as well as closes some of the gaps in the detected shapes.

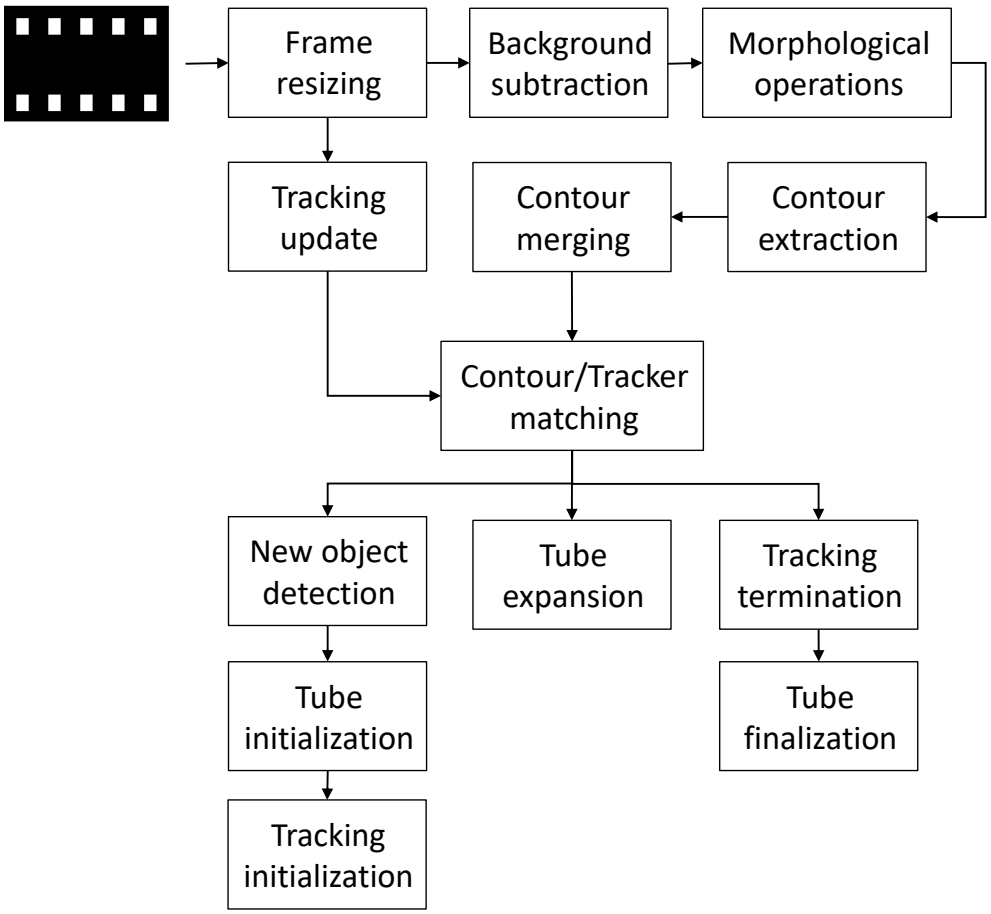

Figure 6.2: Diagram of the analysis pipeline

(c) Contour extraction - Once the foreground mask is refined, individual contours are extracted. Only the first level, outer-most contours are taken into account. This speeds up contour detection as well as fills occasional holes in foreground shapes.

(d) Contour merging - A bounding rectangle is placed around each contour. Should this bounding rectangle fully envelop any smaller contours, these will be merged into the larger contour. This helps reduce the total amount of contours as well as merge multiple contours originating in a single object.

3. Tracking - This portion of the pipeline maintains the identity of a detected object across multiple frames, as described in section 4.2.

(a) Tracking update - A set of all active object trackers is kept until their termination. This set of trackers is updated with each new frame being processed. A tracker update causes the tracker to produce a new estimation of the tracked object's location.

(b) Contour/Tracker matching - In this step, the updated trackers are polled the for current positions of tracked objects. These positions are correlated with contours detected in the first part of the pipeline. Contours having large overlaps with the tracker bounding box are assigned to given tracker. This assigns an identity to a detected contour.

(c) Tracking initialization - With each contour matched to a corresponding tracker, the remaining contours represent new objects, that have not yet been tracked. For these contours a new tracker is initialized and placed amongst the active trackers. This constitutes the detection of a new object and a new tube is initialized as well.

(d) Tube expansion - The matched contours are used to expand their trackers' tubes. This is achieved by the addition of a new slice to the target tube, corresponding to the bounding rectangle of the matched contour. Information about the bounding rectangle, as well as the source frame number, is stored.

(e) Tracking termination - Once a tracker has failed to locate its subject or the ensuing contour matching yielded no results, the tracker's TTL (time to live) counter is decremented. Once it reaches zero, the associated tube's slice count is checked. Should the number of slices be less than the initial TTL threshold, the tube is outright discarded, getting rid of random short anomalies. This counter is replenished on each tracking and matching success.

(f) Tube finalization - When a tracking terminates and passes the slice count check, the associated tube can be finalized. This involves the accumulation of relevant information, such which file the tube originated in or what resolution it was detected at. Once this information is compiled, the tube along with all of its slices is stored in the database and is associated with the input file's entry.

Once the activities are extracted, additional problems must be solved. An approximation of the background image needs to be obtained, as described in section 5.2.2. This can be done for free by the background subtractor class provided by OpenCV. Since the background subtractor already keeps a model of the background, no additional computation is necessary, and the image is simply obtained directly from the subtractor. The image itself is then stored in the database as well, and a reference is added to the particular video file's entry. The image is stored using lossless image compression, since it will be the most important part of the rendered output video. The amount of data necessary for storage of a single frame of a video file is minor and can be safely stored in the database itself.

A background video could be produced and used instead. This would help to preserve natural lighting changes of the source video. For example, the background of a video taken during the sunset will change greatly and the visual quality of the rendered output would benefit from a background video instead of a background image. This however, would add additional overhead in form of the need to store additional background frames sampled at various points of the detection process. I have decided not to take this route and instead rely on a single background image. I believe that it provides enough visual information for the viewer of the final summary to understand the spatial relationships between the displayed activities, albeit at reduced consistency at the seams of the overlaid imagery.

6.3.3 Tube description

A suitable method of tube description is necessary. Three main attributes of a tube's source activity are tracked for this purpose and are generated during the description process. An extracted tube's description consists of the following parts:

- Slices - A collection of the individual detected parts of input frames. These are kept in the form of metadata instead of the actual image data.
- Bag of Words histogram - A histogram describing the tube is used for comparison of individual tubes for the purposes of similarity-based tube retrieval and filtering.
- Hue histogram - A histogram describing the tube's hue distribution for the purposes of color-based tube retrieval and filtering.
- Thumbnail - A single small slice taken from the middle of the activity will be used for the purposes of visual representation of the tube in GUI.

Slices

A slice represents a rectangular part of a single frame of an input video. A rectangle is used instead of a foreground mask, because an enormous number of slices is expected to be stored and their handling needs to be as fast as possible. The source frame number is stored alongside the rectangle. Once the actual content of a slice is needed, it must be loaded from the source file.

Bag of Words histogram

In order to be able to perform a comparison of individual tubes a common descriptor is necessary. For the purposes of this project, the Bag of Words (BoW) model was chosen, as described in section 4.5. Each tube contains a histogram obtained during the description process. A reference to the vocabulary used to generate this histogram is tracked as well.

The BoW vocabulary is trained once using the detected tubes in database. This simplifies training, because no additional training set is necessary, and the trained vocabulary is well suited for the type of videos being processed. During the description of a tube, the latest dictionary will always be used.

Hue histogram

Hue information of the tube's slices is used for evaluation of color distribution of an entire tube. HSV (hue, saturation, value) colorspace is used. The advantage of this color space is the ability to only focus on a single channel, which holds the primary color information. This histogram will be used to allow the user to search for tubes by a specified general color. It requires no special training, and thus only needs to be computed once for each tube, without the need for recalculation in contrast with the BoW histogram.

Training process

A visual vocabulary requires initial training. This needs to be done only once, provided the system already contains a sufficient amount of analyzed tubes. The training process involves traversal of all of the detected tubes stored in the database and computation of their slices' SIFT descriptors. The visual vocabulary is trained as follows:

Step 1: The actual image data of a group of slices is preloaded from the source video file. This helps reduce the latency of on-demand video data access. An adjustable frame-skipping is employed to reduce the amount of computation.

Step 2: A constant number of SIFT descriptors is computed. The number of descriptors used in this project is 128.

Step 3: The computed descriptors are fed into the Bag of Words model trainer. Once all of the descriptors are entered, the vocabulary is trained by k-means clustering, as described in sections 4.5 and 4.4. The number of clusters is set to 32.

This produces the BoW vocabulary which is stored in the database for future description of tubes.

Description process

The description process of a tube consists of the computation of its BoW and hue histograms. Since the first two steps of the description process correspond to the training process and the descriptions need to be retrained after each vocabulary change, it is beneficial to execute the description process alongside the training. This, however, is not a requirement.

Step 1: The image data of the tube being described is loaded the same way as described in the first step of the training process.

Step 2: A constant number of SIFT descriptors is computed on each slice and a closest cluster is found for each of them using a feature matcher. A histogram with the number of bins equal to the vocabulary cluster size is obtained.

Step 3: A hue histogram is calculated. This is achieved by transformation of the slice image data into the HSV colorspace, upon which a hue value histogram is calculated. The values of these histograms are then averaged into a single histogram which describes the hue distribution of the entire tube. The size of this histogram is set to 64.

It is important to mention, that the description process does not always need to be applied to all of the detected data. For example, only newly processed files in the current session can be described, if a vocabulary had already been generated. This will however limit the similarity-based tube retrieval to the given subset of tubes in the session, or rather to tubes described by said vocabulary.

In order to allow for global search of all detected tubes, the entire set needs to be described using the same vocabulary, ideally during the initial analysis of tubes, when all of the image data is accessible.

6.4 Database

A regular relational database will be used for storage of processed tubes as well as various other metadata. A small amount of image data is stored as well. These are comprised of thumbnails and video file backgrounds. The database can be deployed locally or on a remote server. For the purposes of this project a local database will be used.

An alternative solution would be the usage of a file-based storage system. It is possible to store all of the detected metadata in a suitable format like XML. This would allow for better portability of the data and less general overhead. However, the database route allows for easier execution of complicated queries, as well as ensures data consistency without the need for design and implementation of additional input/output systems.

6.4.1 Entities

This section contains an overview of the most important database model entities as well as an ER diagram of the final database (Figure 6.3).

Slice

The Slice database entity fits the description of a general slice defined in 6.3.3. It is the most numerous entity in the database. A slice consists of only its bounding rectangle's coordinates stored as integers and the frame number it was detected in, stored as a long integer.

Tube and Shifted tube

The Tube database entity is the representation of an activity tube. It serves as a parent for a set of Slices. It also contains following properties:

- BoW histogram and Vocabulary reference - The histogram used for similarity comparison, as well as a reference to the vocabulary used to generate this histogram.
- Hue histogram - Hue histogram used for color-based queries. Described in 6.3.3.
- Thumbnail - A small, binary encoded image representing the file for easier management in the GUI.

A Shifted tube serves as a wrapper for a regular tube, adding information about the generated time offset. A Tube can be referenced by multiple Shifted tubes. The starting and ending frame numbers are also stored in a shifted tube to speed up look up processes.

Video File

The Video File database entity represents a single video file. It serves as a parent for a set of Tubes. There are two types of video files stored in the database. The input video files, and the rendered output summaries. These are distinguished by the *IsSummary* binary flag. A video file entry contains the basic information necessary for the summarization process and playback.

Following properties of a video file are tracked:

- Path and Name - These are the basic filesystem variables used to access the physical video file. Should the actual file be renamed or moved to a different directory, these must be updated.
- Width and Height - The dimensions at which the file was processed. These correspond to the resolution mentioned in section 6.3.2.
- Size and Checksum - These are used for identification of duplicate files. The size of files in bytes and an MD5 checksum are used.
- Summary flag - A Boolean value distinguishing input and output files.
- Length - The total running length of the video file stored as ticks.
- FPS - A floating point number describing the framerate of the video file.
- Thumbnail - Equivalent to the tube thumbnail mentioned above.

Summary

The Summary database entity represents the temporal alignment of a set of tubes. It contains a set of Shifted tubes, used for rendering of the output video. It also contains a reference to the output Video File entry, if rendered.

Session

The Session database entity represents a user session of the GUI app. It provides a layer of persistency in the application as well as serves as the initial filter, selecting a subset of files for further processing. It contains information about the creation date of the session. A session may contain multiple Summaries.

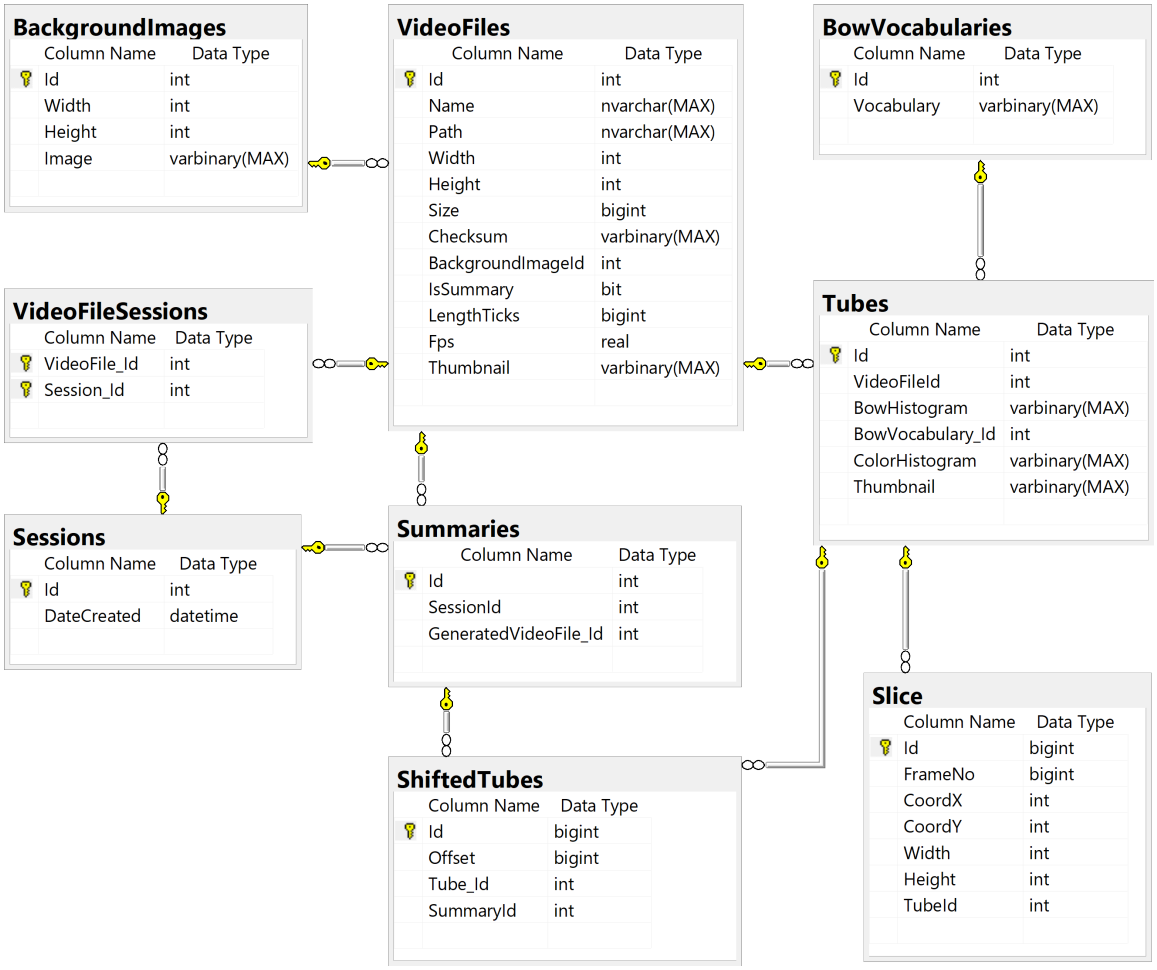

Figure 6.3: Entity Relationship diagram of the designed system

6.5 Query processor

The Query processor is the part of the summarization system responsible for evaluation and application of user queries. With tubes already successfully extracted, analyzed and stored, there is a need to determine which tubes should be displayed and which should be discarded. Some tubes are discarded outright during the initial analysis process. These include short activities, or global scene changes such as sudden camera movement. Because of this, it can be safely assumed that the remaining tubes are „healthy“. A tube is considered healthy when the activity it contains is consistent and without any major flaws. User may also specify queries for filtering and search within the processed activities without the need to produce a resulting summary.

When looking for a specific activity in a video, only a subset of tubes will be of interest while the rest can be ommited from the resulting summary. This act of filtering will be performed in adherence to user defined constraints. This part of the system will communicate with the user using a graphical user interface, described in section 6.7. This interface will allow for specification of said queries, as well as respond to the user's commands in real time, providing filtered results upon specification of individual constraints. The basic

idea behind a query-enabled summary comes from the paper *Webcam Synopsis: Peeking Around the World.* [13]

6.5.1 Queries and Constraints

In order to make the explanations of the various types of filtering clearer, two terms which were up until now used interchangeably need to be distinguished and defined.

- A *Constraint* represents a single condition that must be satisfied by a filtered tube. It is the basic building block used by the user to define *queries*. Constraints may be of various types. Should a constraint of the same type be defined multiple times, the operation of logical disjunction will be applied to combine them into a single constraint of the same type. This means that at least one constraint of a given type must be satisfied.

- A *Query* represents the entire set of constraints defined by the user, combined into a single condition that is required to be met by all of the filtered tubes. The logical operation of conjunction is applied to the individual constraints constituting the query. This effectively means that each type of defined constraints must be satisfied.

For the purposes of this text, I have identified these three basic *categories* the constraints can be divided into:

- Temporal - Constraints falling into this category are based around the restriction of the time dimension.

- Spatial - Since the temporal constraints restrict the 3D tube space along its time axis, the spatial constraints restrict the tubes in the two remaining dimensions. This means that spatial constraints revolve around filtering of the tubes according to their slices' sizes and positions.

- Visual - The remaining attributes of a tube can be described as visual. These are based on the actual graphical *content* of the slices comprising the tubes.

A single constraint *category* may contain multiple *types* of constraints. A constraint *category* only serves as a conceptual unit used for the purposes of this text, while the *type* of a query is an actual attribute used in the filtering process, conforming to the definition of *queries* and *constraints* laid out above.

In the following paragraphs, each constraint category will be explored, defining its individual constraint type in terms of its variables and the description of the evaluation process.

6.5.2 Temporal constraints

A single type of a temporal constraint will be available in the system. The temporal constraint will have these three components:

- Video file context - An input video file on which the constraint will be applied must be specified. Enforcement of a time constraint on all files in the current session would only make sense if the system had information about the actual time of the day during the recording of the video, which is outside the scope of this project.

- Starting point - A point in the selected video's timeline must be specified.
- Ending point - A point in time specifying the end of the time-span of interest.

Once these three values are obtained from the user, the constraint can be applied. Should the user need to specify multiple time spans within the same video, additional instances of time constraints can be added and will be combined using the logical operations described above.

Evaluation of a temporal constraint

The temporal filtering is applied as follows:

Step 1: All tubes originating in file specified, which currently conform to any preexisting filters, will be checked.

Step 2: The starting and ending frame of each tube will be determined based on its slices.

Step 3: A tube will be deemed satisfactory if its ending frame occurs after the specified starting point and its starting frame occurs before the ending point of the period.

An example of a temporal constraint can read as follows.

„Show, all activities from video A, occurring between timestamps 00:07:00 and 00:12:00."

6.5.3 Spatial constraints

This category of constraints deals with the individual positions and sizes of tube's slices in the X and Y dimensions of the video. Two types of spatial queries will be present in the system.

- Region of interest - An area of interest is specified. Activities detected outside of this area can be safely omitted.
- Direction - A direction is specified. Activities moving in this general direction must be included in the filtered results.

Following variables need to be specified by the user in order to perform filtering. For the ROI constraint, a rectangular region of interest must be specified by the user through use of the GUI. In case of the direction constraint, a directional arrow will be drawn by the user. Only the angle of this arrow will be taken into account, with its length being purely cosmetic.

Evaluation of a region of interest constraint

The region of interest constraint is applied as follows:

Step 1: All tubes originating in any file of the current session, which currently conform to any preexisting filters, will be checked.

Step 2: Each tube's slices' bounding rectangles will be checked for an overlap with the specified rectangular region of interest

Step 3: A tube will be deemed satisfactory if any of its slices have at any point overlapped with the ROI.

Evaluation of a directional constraint

The directional constraint is applied as follows:

Step 1: All tubes originating in any file of the current session, which currently conform to any preexisting filters, will be checked.

Step 2: The starting and ending spatial positions of each tube will be determined based on the positions of their starting and ending slices.

Step 3: A vector will be obtained from these positions.

Step 4: A vector will be obtained from the user drawn arrow.

Step 5: A tube will be deemed satisfactory if the angular difference of the two vectors will be below a certain tolerance value.

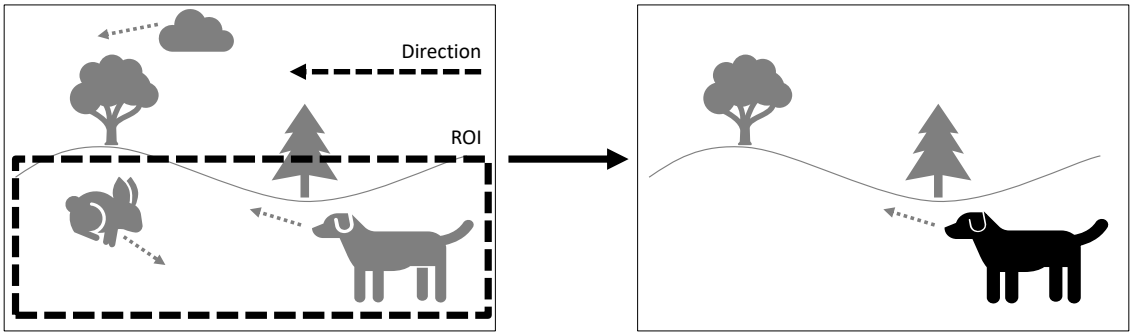

Figure 6.4: An example of simultaneous application of both types spatial constraints. Only activities detected within the region of interest (ROI) and matching the specified direction will appear in the summarized output.

6.5.4 Visual constraints

The final category of constraints is the most complex. Two types of visual queries will be available in the designed system. They are the following:

- Similarity - The retrieval of tubes similar to a specified tube.
- Color - The retrieval of tubes of a given color.

A single input is required for the evaluation of each these constraints. In case of similarity, a source tube must be selected. As for the color constraint, a target hue needs to be specified using a graphical hue picker.

Evaluation of a similarity constraint

Step 1: All tubes originating in any file of the current session, which currently conform to any preexisting filters, will be checked.

Step 2: A BoW histogram will be loaded for the selected tube.

Step 3: Each tube's own BoW histogram will be compared with said histogram, by the means of cosine similarity.

Step 4: The resulting similarity measure will be transformed into a percentual range of 0-100.

Step 5: A tube will be deemed satisfactory, if the calculated similarity surpasses a set percentual threshold.

The details regarding the Bag of Words model and histogram were already discussed in the sections 4.5 and 6.3.3 respectively. An example of a similarity constraint can read as follows.

„Show, all activities similar to activity A.“

Evaluation of a color constraint

Step 1: All tubes originating in any file of the current session, which currently conform to any preexisting filters, will be checked.

Step 2: A hue histogram will be generated from the selected hue value combined with a gaussian filter.

Step 3: Each tube's hue histogram will be compared with said histogram, by the means of cosine similarity.

Step 4: The resulting similarity measure will be transformed into a percentual range of 0-100.

Step 5: A tube will be deemed satisfactory, if the calculated similarity surpasses a set percentual threshold.

The details regarding the hue histogram were already discussed in the sections 6.3.3. An example of a similarity constraint can read as follows.

„Show, all activities containing primarily red objects.“

6.6 Output assembler

The output assembler takes care of the final steps of the summarization process. Having determined which tubes should appear in the output summary, a way to transform these inputs into the final output video needs to be devised. This task consists of phases that are independent and time-consuming enough to be decoupled into two individual processes.

These are the following:

- Summary generation - This phase produces the final temporal rearrangement of relevant tubes.
- Output video rendering - Once the tube arrangement is generated, the physical output file can be rendered.

Each process can be launched independently, in order to allow for a quick review and prospective regeneration of a generated summary without the need for the costly rendering process to take place each time. The generated arrangement will be visualized in the graphical using interface using bounding rectagles without the actual contents of their slices.

6.6.1 Summary generation

The final subset of tubes needs to be rearranged in such way, that the resulting arrangement takes up a significantly smaller amount of time. The two spatial dimensions are fixed, because the tubes are required to maintain their spatial relationships as well as correspondence to the background. This leaves us with one dimension, the time. Taking this into account, it is possible to define the aim of the summary generation step, as a search for each individual tube's time offset. This time offset will represent a temporal position in the output video file, at which the given tube will be projected into the output. This needs to be done in such way, that no tubes are interfering with any other tubes.

A first-fit algorithm is employed for the purposes of offset determination. A simplified description of the algorithm goes as follows:

Step 1: The desired tubes are sorted primarily by their input files, and secondarily by their detection time.

Step 2: A set of output frame *reservations* is established. These reservations consist of a collection of rectangles already occupying space in a given output frame.

Step 3: Current tube's slices are iterated over and existing reservations are checked for occupancy of given slice's bounding rectangle. This can be efficiently done using a simple rectangle coordinate comparison.

Step 4: If a collision is detected, the tube's offset is increased, and Step 3 is taken again.

Step 5: If no collision occurs, output reservations are placed for current tube's slices, and the final offset for the tube is produced.

Step 6: If no tubes remain, the processing is finished. Otherwise, the next tube is handled returning to Step 2.

Once all of the offsets are determined, a new *Summary* database entity, described in section 6.4, will be generated, complete with a set of *Shifted tubes*. These will be stored in the database for later review or commencement of the rendering process.

6.6.2 Output video rendering

Once the offsets are determined, the actual summary video can be generated. Since only the slice metadata is kept in the database, the source video files will need to be accessed throughout the rendering process. Simple sequential drawing of the tubes cannot be employed in the rendering process, because the nature of the output requires it to be written frame by frame instead of tube by tube. For this reason, I have devised the following method of output summary rendering.

Rendering will be done in *chunks*. A *chunk* is a sequential subset of the output frames of constant size. Each chunk is defined by its starting and ending frame. The smallest possible chunk is a single frame long, in which case the rendering process is equivalent to simple frame by frame rendering. The chunk-based rendering process is comprised of the following steps:

Step 1: Relevant tubes are identified. This is done by checking all of the desired tubes for a temporal overlap with the chunk's time period.

Step 2: Metadata of the relevant tubes' slices is loaded from the database.

Step 3: The actual image data for these slices is buffered from the source video files.

Step 4: A frame of the output video file is initialized with the default background image.

Step 5: Each tube's slices, where the combination of slice frame number and tube offset matches the current output frame number, are projected onto the output frame.

Step 6: When all relevant slices are projected, the next frame is processed from Step 4.

Step 7: When all of the chunk's frames are rendered, the chunk is written into the output file and all resources allocated by the chunk are released. The next chunk is processed from Step 1.

Step 8: When all chunks are rendered, a new entry describing the generated video file is added to the database. A reference to this entry is created under the current *summary* entity's *generated video file* property, as defined in section 6.4.

The usage of chunks is a compromise between the speed of rendering and memory requirements of the process. As the size of a single chunk increases, the memory requirements rise as well. Large chunk size however results in less frequent input video file reads, which leads to a speed up of the actual rendering of individual slices.

Another advantage of this kind of compartmentalization of the output rendering is the ability to easily parallelize the process. It is even possible to distribute the task of rendering individual chunks between multiple rendering servers and assemble the final output after all chunks have been rendered. This however, is beyond the scope of this project, meaning that only the basic, sequential chunk rendering will be employed during the implementation.

6.7 GUI application

With the individual parts of the system laid out, the design of the graphical user interface can be defined. The user interface will take the form of a regular desktop application, centered around the mouse as the primary source of input.

The application will facilitate access to the following features:

- Video file management
  - Source video playback - The user will have the ability to browse and play the input videos. Dynamic annotations will be used to give the detected tubes a context and a layer of interactivity.
  - Introduction of new input files to the system - The user will be allowed to start the analysis process from within the application itself without the need to use the console interface.
  - Summarization output playback - The user will have the ability to play the generated summaries without leaving the application.
- Tube management
  - Tube playback - The user will have the ability to browse and display an interactive representation of the detected tubes. Dynamic annotations will be used.
  - Similar tube search - Any given tube will have the capability of being used as the basis for a similarity search.
  - Tube description generation and training - The user will be able to launch the tube description process described in section 6.3.3.
- Session management
  - Session creation - The user will have the ability to create a new session, selecting the desired files.
  - Session selection - Previously created sessions will have the ability to be loaded and worked with.
- Summary management
  - Summary generation - The user will have the ability to generate and visualize the arrangement of tubes described in section 6.6.1.
  - Summary rendering - The user will have the ability render the video outputs of the previously generated video summaries.
- Entity deletion - The user will be provided by means of deleting any of the mentioned entities from within the application.
- Query specification - The user will be able to enter individual constraints.
- Progress communication - The user will be informed about the progress of long running tasks.

6.7.1 Mockups and GUI description

This section contains mockups of the user interface as well as the description of its key components. The description contains multiple instances of *italicized* words. In this section, they are used to refer to specific UI elements, pertaining to the mockups. I have tried to keep the user interface as simple as possible, without hindering any functionality. The final design consists of a single main window, visible in Figure 6.5, and an initial modal dialog

prompting the user to either select or create a new session. The mockup of the dialog can be seen in Figure 6.6. Generic message boxes and confirmation dialogs will be used where necessary and are not important enough to be warrant a detailed description in this text.

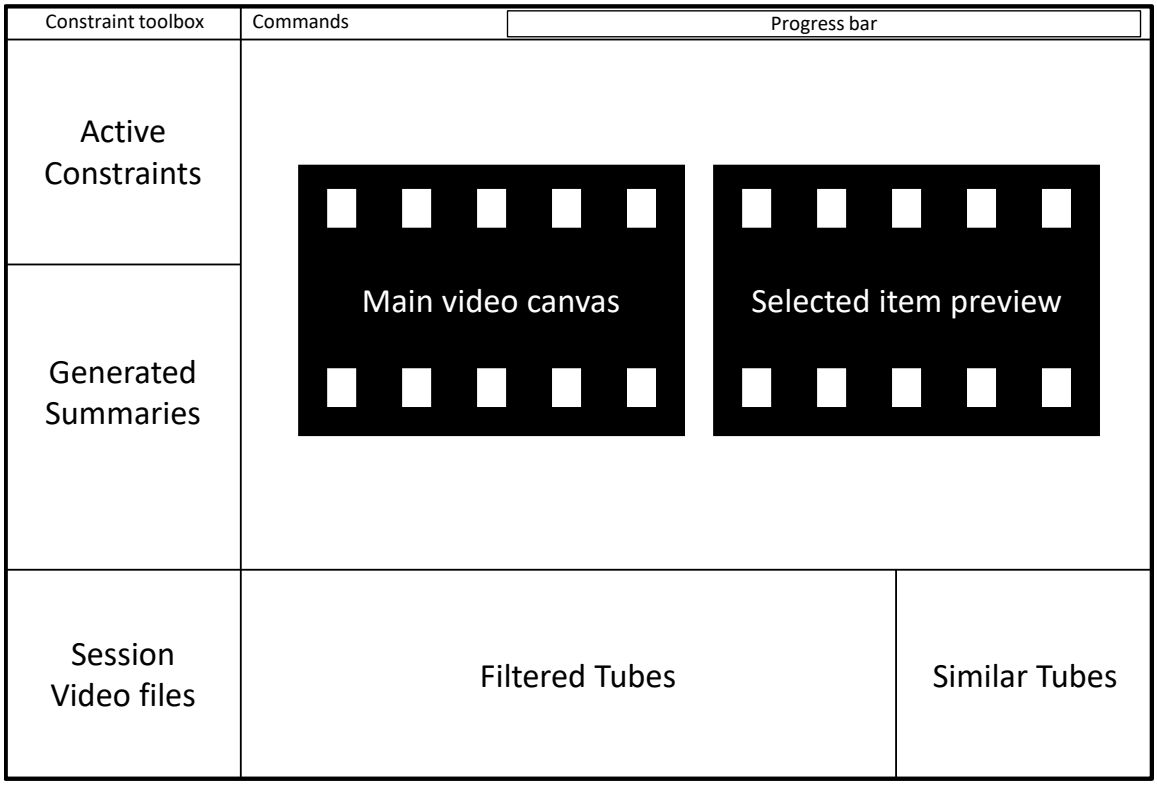

Figure 6.5: A mockup of main application window

Center panel

The centerpiece of the application is a pair of video players. Two players are used for easier previewing of selected tubes. The *Main video canvas* will be used for playback of the active source video, or the final output video. It will also serve two additional purposes aside from video playback. It bears the word *canvas* in its name because it serves as the input method for the spatial constraints defined in section 6.5.3. The user will draw their constraints directly onto the video canvas itself. The other type of drawing that will be done on this canvas is drawing of the context annotations for displayed tubes by the UI itself. A bounding rectangle will be drawn for each slice, displaying the time in the source video at which the activity occurred. This dynamic annotation will be clickable and will provide the user with a preview of the selected activity.

This brings us to the second video component of the center part, the *Selected item preview* pane. This second video pane will display any selected tube in an infinite loop. This will let the user view different tubes without losing their position in the main video player. Overlays will also be drawn in this player, however they will not be interactive.

Both video players will have standard video controls, comprised of a play/pause toggle button, a timeline seek bar and length counter. Additionally, the players will be zoomable

using the mouse scroll wheel. The speed of the video playback will be adjustable by pressing of the plus (+) and minus (-) keyboard keys.

Bottom panel

The bottom part of the UI will be divided into three parts. The first, leftmost part will display the current *Session video files*. Upon clicking a video from this list, the video will be displayed on the *Main video canvas*, and will be overlaid with the interactive annotations described above.

The next section is the second largest part of the entire UI. This is because it hosts the list of *Filtered tubes*, conforming to the currently applied constraints. Since a large number of tubes is expected to be present in the system, pagination will be applied to this list. Clicking an item in this list will cause the *Selected item preview* video pane to display the given tube, complete with a dynamic annotation. A right click will display a context menu, letting the user search for similar tubes.

The last part of the bottom panel is reserved for the results of similar tube search. This list is similar to the *Filtered tube* list, with the difference of additional percentual information regarding the tube similarity being displayed. Clicking an item in this list will also cause the *Selected item preview* to play corresponding tube.

Left panel

The left part of the UI contains two main parts. The upper-most part is the list of *Active constraints*. The specification of new constraints will be initiated by clicking the desired type of constraint in the *Constraint toolbox* located above this list. Depending on the type of selected constraint, the definition process will be started. An exception to this rule comes in the form on the Similarity constraint, which is defined via the context menu of a tube.

Spatial constraints will be drawn directly on the video canvas. Selecting an existing spatial constraint in the list will highlight it on the *Main video canvas*.

Temporal constraints will be defined directly in the list item of the constraint using buttons setting the required timestamps to the current position of the *Main video canvas* playback and a combo box for the selection of a relevant video file.

Color constraints will also be editable directly from the list item. This will be handled by a single hue seek bar.

Additionally, a reset button will be provided to quickly remove all of the defined constraints.

Bellow the *Active constraints* list lies the *Generated summaries* panel. This list contains all summaries generated in the current session, by the process described in section 6.6.1. An item in this list will display the number of tubes contained in given summary, as well as the rendering state of this summary. The rendering state can range from *Not rendered*, *Rendering* to *Rendered*. If the summary is *Not rendered*, clicking it will display only the interactive annotations representing the shifted tubes, visualized on a static background image. In order to be able to display the actual image data of activities in the summary, the output video must be rendered. This will be done by clicking a button on the summary item itself. All playback will be halted until the video is rendered. When a *Rendered* summary item is clicked, the rendered output video will be displayed on the *Main video canvas*, along with the interactive overlays described before. The summary item will also contain a button taking the user straight to the generated file in the operating system's file explorer window.

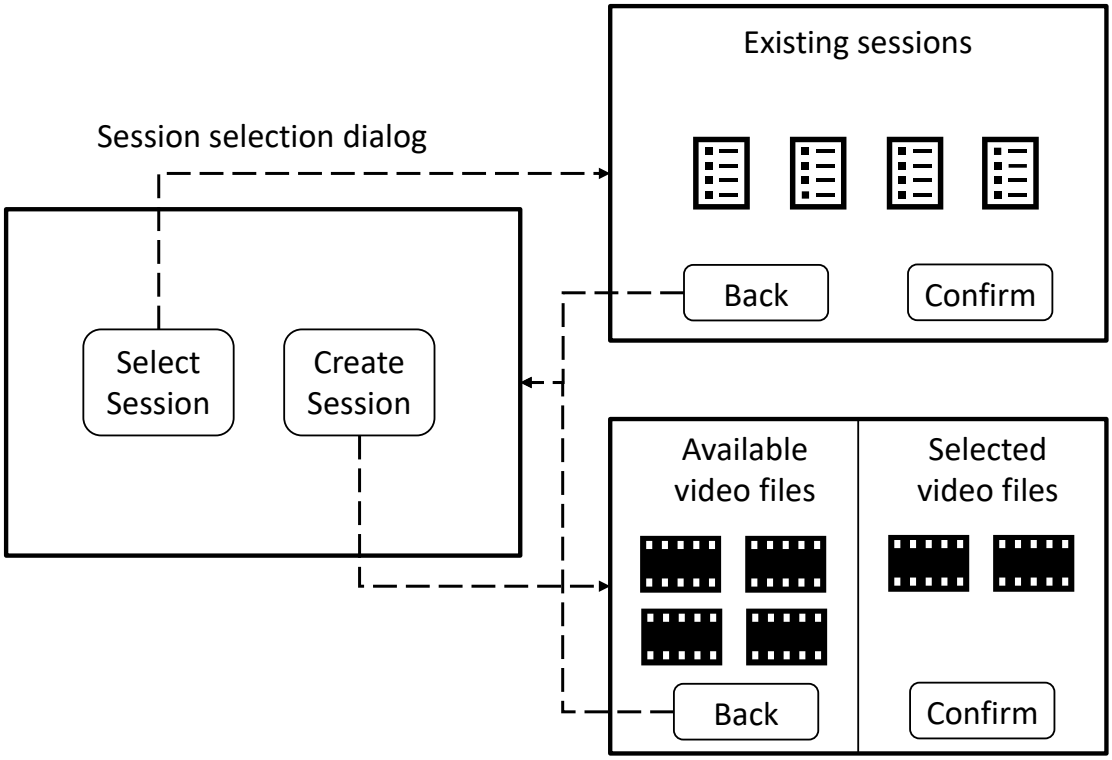

Figure 6.6: A mockup of the *Session selection dialog* window.

Top panel

The last part of the main UI screen is the top panel. This panel will contain remaining commands. These are the launching button for the *Session selection dialog* window, a button for analysis of a new video, a button for retraining of tube descriptions and a cancellation button for long operations. The rest of this panel is occupied by the progress bar. The progress bar will also contain status messages for long operations.

Session selection dialog

This modal dialog lets the user select between two options of session selection. The user can either choose to load an existing session from the database, leading them to the actual session selection, or they can opt for creation of a new session. This will take the user to the session creation part of the dialog, where a subset of all video is selected using two lists of videos. Upon confirmation of either part of the dialog, it is closed, and the selected session is loaded.

Chapter 7

Implementation

The aim of this chapter is to highlight some of the more interesting aspects of the implementation process. I discuss my choices and describe the challenges faced during this period.

7.1 Choice of tools and technologies

After the initial research and definition of the individual subsystems described in chapter 6, I set out to implement the described system. The first step in the process was the selection of a development platform, suitable tools and frameworks. Initially, I chose the following:

- Microsoft Windows 10 as target and development platform.
- OpenCV 3.3 as the core library for the majority of computer vision tasks.
- Python 3.6 as the implementation language.

Microsoft Windows was chosen because it is an operating system that I use on a daily basis and one that I feel the most comfortable with. The OpenCV *(Open Source Computer Vision)* library is an obvious choice for the majority of computer vision related projects. This open source, cross-platform library would provide me with a wide variety of important tools accessible from C/C++, Java and Python [19].

Finally, Python programing language was selected for its ease of prototyping and ability to quickly implement and test out ideas. I was fully aware that a C/C++ implementation would prove to be more efficient, nevertheless I chose to sacrifice some of the final product's speed for faster development. This combination of Python and OpenCV would also make the final product quite portable.

However, after the implementation of primary parts of the *Input analyzer*, described in section 6.3, I started to realize that I may not be as proficient in Python as I have initially thought myself to be, so I started looking for an alternative.

After researching the availability of various OpenCV wrappers, I have reconsidered my initial choice of tools as follows:

- C# and .NET Framework 4.6.1 were chosen as the language and framework.
- Emgu CV 3.4 as the C# wrapper for OpenCV 3.4.
- Windows Presentation Foundation (WPF) was used for the GUI of the system.

- Microsoft SQL Server 2017 was used as the backing database system.
- Entity Framework 6.2 was used for database management and communication.
- Microsoft Visual Studio 2017 was used as the main implementation IDE.
- Microsoft Visual Studio Team Services were used for version control.

This combination of tools, with the exception of WPF, is one that I am familiar with. This has allowed me to quickly implement the system I had designed, without the need to focus on other issues that come with learning a new language or framework. In addition to the items listed above, a few minor libraries mentioned throughout this chapter were obtained using the *NuGet* package system.

7.2 Solution structure

The entire system was implemented as a single *Solution*. A *Solution* is a Visual Studio concept representing a complex system consisting of several *Projects*. A *Project* is a self-contained part of the codebase, usually producing single output, in the form of a library or an application. I have divided the designed system into the following projects:

Summary.Backend
: This is the core of the solution containing the majority of individual subsystems, described in section 6.1. The output of this project is a class library.

Summary.DAL
: This project represents the data access layer (DAL) of the system. It provides a facade for each part of the system that needs to interface with the database described in section 6.4. The output of this project is a class library.

Summary.Analyzer
: This project represents the standalone input analyzer application describe in section 6.3. The output of this project is a console application.

Summary.App
: This project represents the man GUI application described in section 6.7. The output of this project if a Windows application.

7.3 Implementation of individual components

This section contains descriptions of the individual components of the implemented system.

7.3.1 Backend

The *Backend* project implements all of the important subsystems of the summarization process. It references the DAL layer for the purposes of database access.

Input analyzer
: This part is comprised of individual subsystems, corresponding to the proposed design from section 6.3. The *SmartThreadpool*[1] library is used to move database access to multiple different threads. These are the components:

Frame provider is responsible for the reading of the input videos. In my implementation I have decided to use the regular RGB image representation consisting of three channels with the depth of a single byte each. An instance of a frame provider is initialized with the path of the desired file, and the target working resolution. The actual reading of the file's frames is facilitated by the OpenCV class *Capture*. The frame provider also offers buffered reading and preloading of a tube's slices' image data, utilized in the process described in section 6.6.1.

Mask extractor implements the foreground extraction and refinement steps described in section 6.3.2. The *BackgroundSubtractorMOG2* OpenCV class is used for the initial background subtraction.

Contour extractor detects and filters contours of the extracted mask. This is done using the OpenCV method *FindContours*.

Tube detector handles the tracking and recording processes of individual tubes. OpenCV class *TrackerMedianFlow* is used for the actual tracking.

Tube processor oversees tube finalization and description. It also implements the training processes described in section 6.3.3. Training of the Bag of Words model is handled by the OpenCV class *BOWKMeansTrainer*, while the features used are implemented by the OpenCV class *SIFT*. The matching of the keypoints is done using the *FlannBasedMatcher* class.

Query processor

This subsystem, as designed in 6.5, handles the retrieval of similar tubes and application of the individual constraints described in section 6.5. All of the implemented constraints are an extension of an abstract *TubeConstraint* class. This class defines the type of constraint and a method which determines whether or not a tube satisfies this constraint. This allows for quick addition and implementation of new tube constraint types.

Summary assembler

The final part of the backend project handles the tasks described in section 6.6.1. It facilitates the generation of time offsets as well as the actual rendering process. An ability to locate the generated file is present. This is achieved by launching the *Windows Explorer* in a new process targeted at the output file.

7.3.2 Data access layer

The implementation of the database communications was largely done using the *Entity Framework* library. This library offers various abstractions of the database system. It also facilitates the creation of the entire database schema on a code-first basis. This was utilized during the design of the database. The individual entities described in section 6.4 were implemented in the form of model classes located in the namespace *Summar.DAL.Model*. An application database context was created, defining sets of these model classes. These sets were then transformed in a database schema by the framework itself. Subsequent changes to the model classes were tracked and applied using database migrations.

A number of *Facade* classes was implemented, corresponding to each part of the system requiring database access. Each facade exposes a subset of database tasks necessary for the given part of the system.

For the purposes of communication between the DAL and other layers of the system, a set of data transfer object classes, or DTOs, was created, resembling the model classes. These are purpose-built, data-only classes used in specific places of the individual parts of the system. The DTOs are used as inputs and outputs of the facade methods. The model classes are never exposed beyond the scope of this layer.

7.3.3 Analyzer console application

This is the simplest part of the system consisting of a single console application providing an interface for the Input analyzer subsystem implemented in the *Backend* class library described in the previous section. The purpose of this application is to allow the user automated launching of the input analysis without the need for a GUI application, or further interactions. The *CommandLineParser* [10] library is used for parsing of the input arguments.

7.3.4 Windows Application

The main application provides access to all of the system's features. It is one of the most complex parts of the system. The application was implemented using the Model-View-ViewModel (MVVM) pattern. This means that large parts of the code responsible for synchronization of the user interface and data structures were not necessary to be implemented, as this task is handled by viewmodel bindings and commands. A library called *QuickConverter* was used to reduce the number of necessary value converters. This library allows for inline specification of simple converters directly in the WPF bindings. [9] For example, one could bind a Visibility property to a numeric value using an expression, without the need for a converter class. Various custom WPF controls were created for this project, with the most important being the *Video canvas* described in the section 6.7.

Video canvas

This control primarily serves as a video player. My initial implementation relied on the built in WPF control called the *MediaElement*. This however proved to be a terrible choice, as the control is practically unusable with today's video formats unless the operating system is carefully set up with the necessary codecs. I have spent multiple days trying to get this control to work reliably with regular H.264 files. I've decided to go the third party route and look for an external library for this purpose. I found the library *WPFMediaKit* which initially appeared to provide support for multiple current formats, but in the end still relied on the Windows's DirectShow filters. Finally I have discovered the *Meta Vlc* library. This library is based on the open source *libVLC* library, which powers the popular, codec-less video player *VLC*. This fulfilled all my needs for video playback.

The second problem that needed to be solved regarding the video canvas was need for a custom drawing capability. I have decided to implement my own version of a drawing control, since I only needed to be able to draw two shapes; a rectangle and an arrow. In order to create a WPF control capable of this, I had to break some of the MVVM pattern's rules, and revert to a more basic, event driven design within this control. The resulting control's interaction is implemented using mouse events and allows for user drawing of required shapes.

The third problem is the drawing of dynamic annotation overlays. This is achieved using regular updates of the video canvas's visual tree upon a change in the video position.

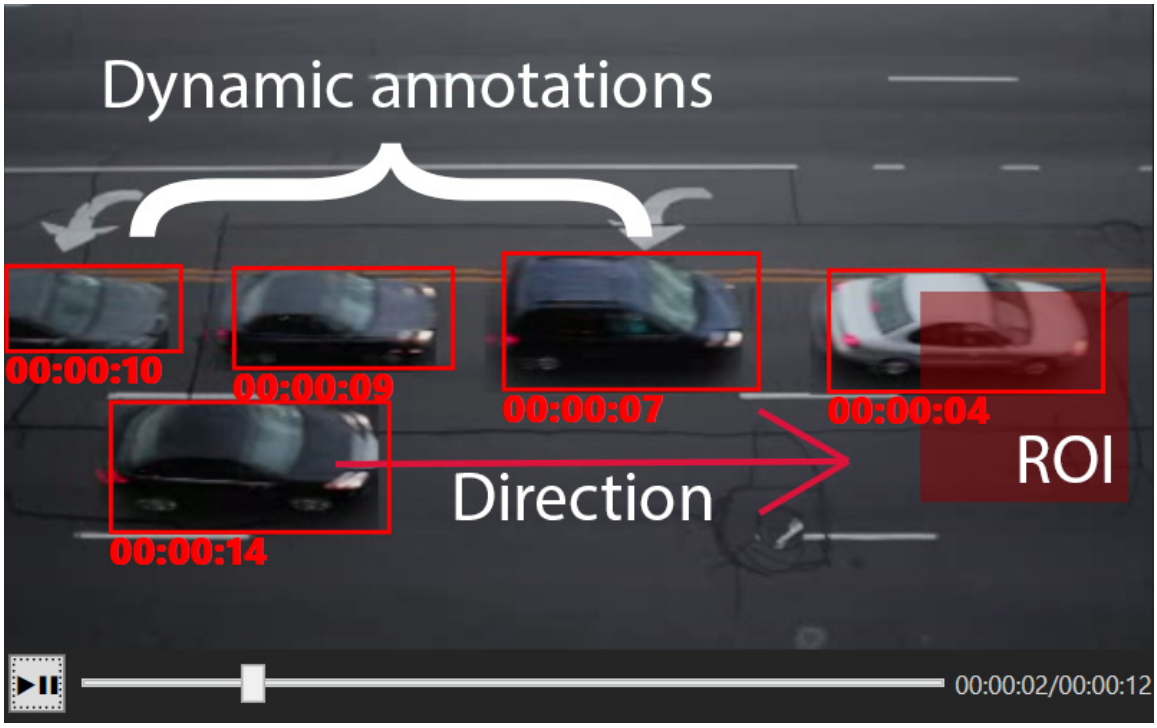

Figure 7.1: A screenshot of the implemented video canvas displaying a summarized video. Direction and ROI constraints are visible as in this image, along with multiple instances of the dynamic annotations of individual tubes. The timestamps below each annotation correspond to the source video's time of the activity.

The VLC library caused a bit of a problem here, with a fixed rate of event updates which is locked at around 250ms. I was able to successfully bypass this limitation through the creation of a separate timer, ticking at the speed the of video file's framerate. This lets me redraw the annotations with each new frame. The VLC position update events are still used for synchronization of the annotation timer. Upon each redraw, relevant tubes are determined based on the current frame number.

Since the video playback may contain a large number of tubes, it is not feasible to load all of these at once. A memory cache is employed instead, storing the previously loaded tubes, until a memory shortage occurs. Upon a cache miss, the cache loads the requested tube from the database. Once the relevant tube is determined and obtained, the current slice is located, and a corresponding *Tracker overlay* control instance is initialized and added to children of the video canvas. The tracker overlay is a clickable rectangle corresponding to a tube's bounding box and displays the actual timestamp of the source video frame it was detected in.

Should a background image be used instead of a background video, as exemplified in the case of a summary preview preceding the rendering described in section 6.7.1, only the timer will be used for the annotation drawing.

In order to simplify the management of the video canvas playback, a *PlaybackService* is implemented as a singleton. This service allows for changes of media and playing state from any part of the UI code.

Multiple instances of video canvas control can be used concurrently, as evidenced by the presence of two video players in the design. All of the elements described above can be seen in Figure 7.1.

Background tasks

Multiple long running tasks are present in the system. For this purpose, a progress bar and an option of cancellation was necessary to be implemented. I have implemented an extended version of a basic background worker, with the ability to post status messages in addition to percentual value regarding the progress. Each viewmodel which contains a progress bar is derived from a base *WorkerViewmodel*, which provides the necessary properties and methods for this feature. Cancellation is implemented using the regular cancellation token available in the extended worker.

Chapter 8

Results

In this chapter I present capabilities of the system I have designed and implemented. This is done through a set of walkthroughs, covering main features of the system. Next, I provide measurements and assess the results. Finally I discuss a list of shortcomings I have identified in my solution.

8.1 Capabilities of the implemented system

Since no type of input videos was explicitly specified in the formal requirements for this project, I have decided to focus the capabilities of the system on summarization of traffic surveillance data. This has allowed me to make assumptions about the size and general predictability of the movement of objects. The input analysis process was designed around these assumptions. This primarily holds true for the selection of tracking solution, which works well for this type of videos as described in section 4.2. Adjustments of the analysis phase would need to be carried out, should the type of processed videos change dramatically. This however is a fairly manageable task, since the rest of the system is decoupled from the analyzer, which means that the analyzer can be easily swapped for a different implementation, fine-tuned to other types of input videos.

With these clarifications out of the way, I can safely state, that I have successfully managed to implement all of the features designed in chapter 6. The implemented application allows for analysis, filtering, similarity-based retrieval and most importantly summarization of the input videos, using a graphical user interface.

8.2 Usage examples

In this section, multiple examples covering different features of the application are demonstrated. The best way to demonstrate the system's capabilities is to show it in action. Since the nature of the content does not translate well into plain text, screenshots of the system, coupled with short descriptions are used.

8.2.1 Example 1

The aim of the first example is to demonstrate the process of session selection and creation. Input videos must be present in the system prior to attempting the steps described in this example. The input analysis does not have a dedicated example of its own, since minimal user interaction is required.

This example covers the following functionality:

- Creation of a new session
- Selection of an existing session

When the application is launched, the user is presented with the session selection dialog (Figure 8.1). The user specifies whether an existing session will be selected or a new one will be created. After this is determined, the relevant part of the dialog is revealed. These can be seen in Figures 8.2 and 8.3. The session creation dialog lets the user add any combination of compatible videos. Upon confirmation, the given session is loaded in the main window.

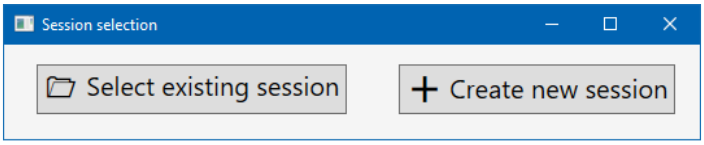

Figure 8.1: A resized screenshot of the session selection dialog from Example 1

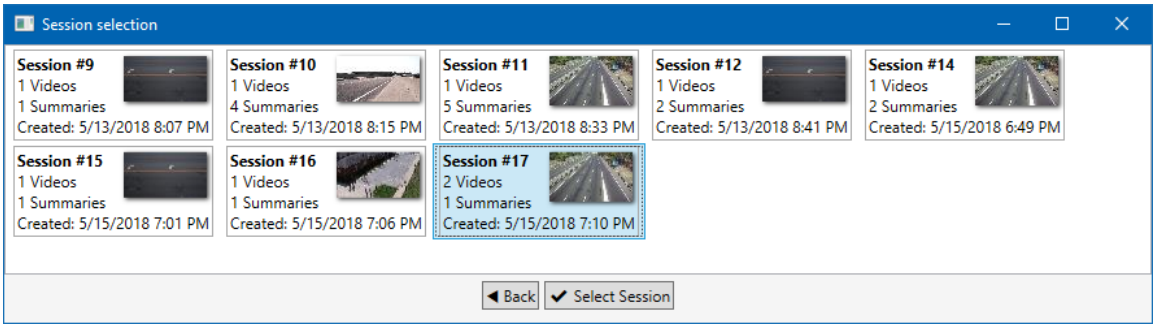

Figure 8.2: A resized screenshot of the session selection dialog from Example 1

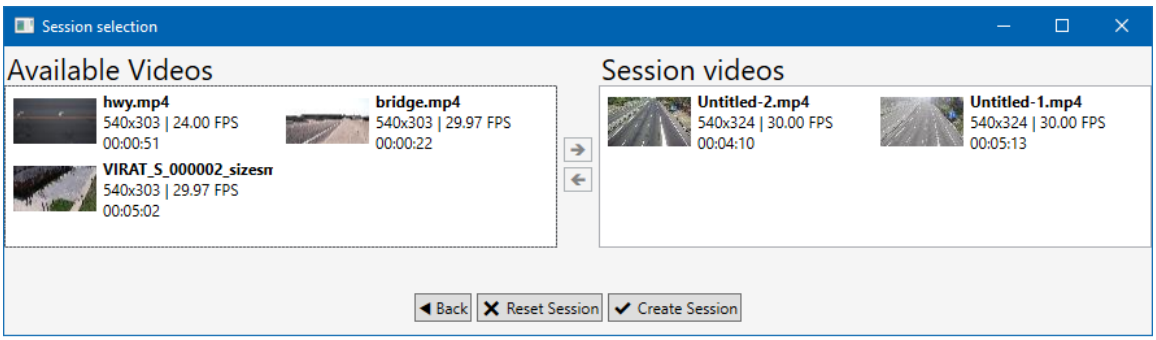

Figure 8.3: A resized screenshot of the session creation dialog from Example 1

8.2.2 Example 2

The aim of this example is to generate a video summary of all red vehicles appearing in specified lanes. This example covers the following functionality:

- Generation of a single summary video from multiple source sequences
- Color constraint application
- Region of interest constraint application

Walkthrough

A session with multiple input videos is selected as show in previous example. Two constraints are specified; a color constraint with red hue is specified and a region of interest constraint is drawn to cover the two leftmost lanes of the highway. This is visible in the top left part of Figure 8.5. A summary is generated using the *Generate summary button*. This results in a preview of the summary visible in Figure 8.4. An output video is produced using the provided button, which results in the final summary being rendered and played back within the application, along with interactive annotations. This is visible in Figure. 8.5

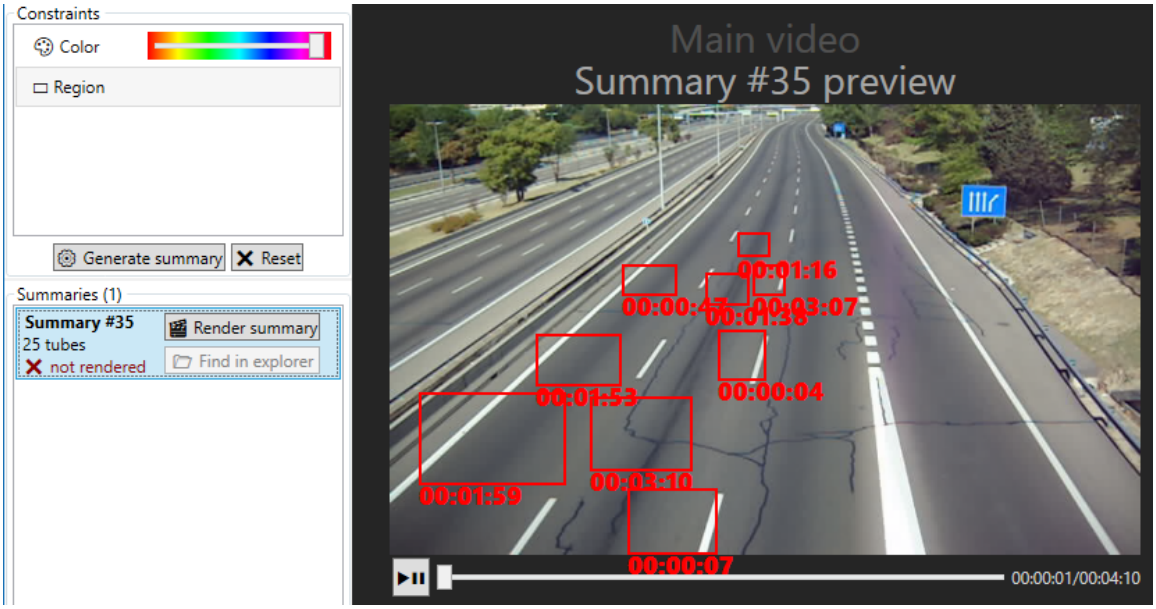

Figure 8.4: A screenshot of a preview of generated summary described in Example 2

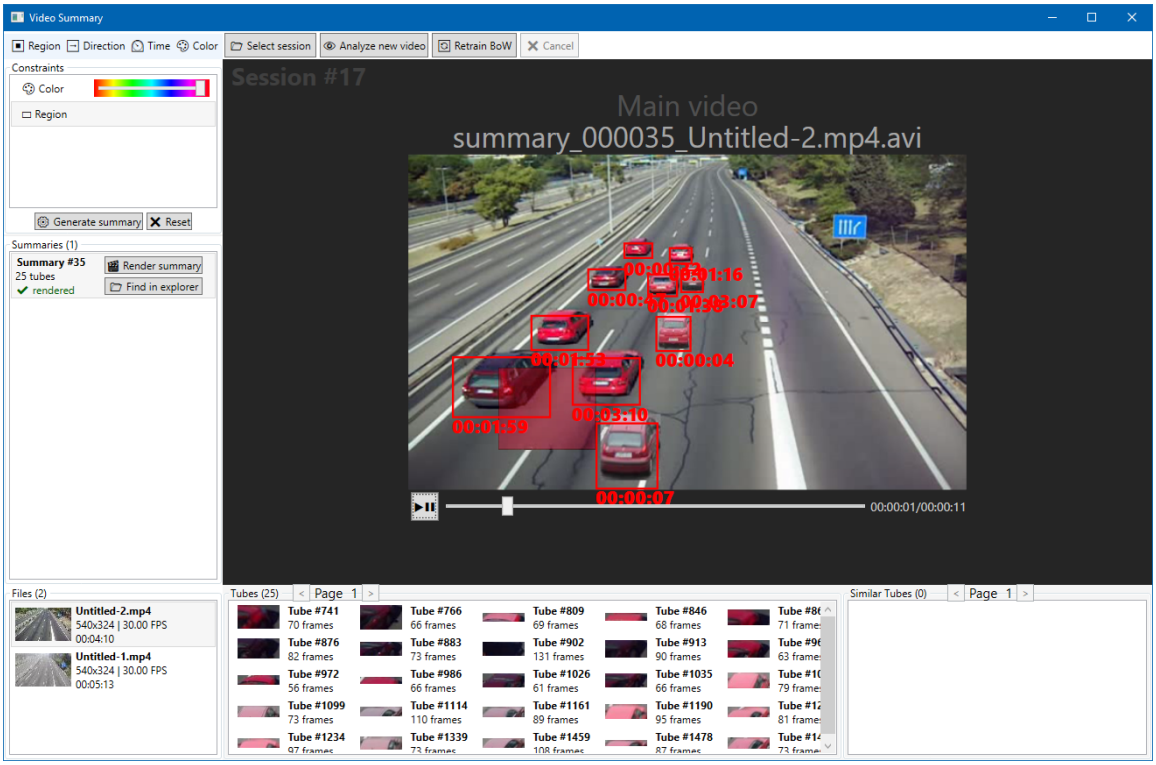

Figure 8.5: A screenshot of a result of video summarization described in Example 2

8.2.3 Example 3

The aim of this example is to generate a video summary of all vehicles going from right to left, in the second half of the video. This example covers the following functionality:

- Generation of a summary video from single source
- Temporal constraint application
- Directional constraint application

Walkthrough

A session with a single file is created. This file contains cars going in two directions. Two constraints are specified; a directional constraint going from right to left, and a temporal constraint restricting the video to its second half. A summary is generated following the same steps as Example 1. The resulting video contains only activities matching the defined constraints, as is evident from Figure 8.6.

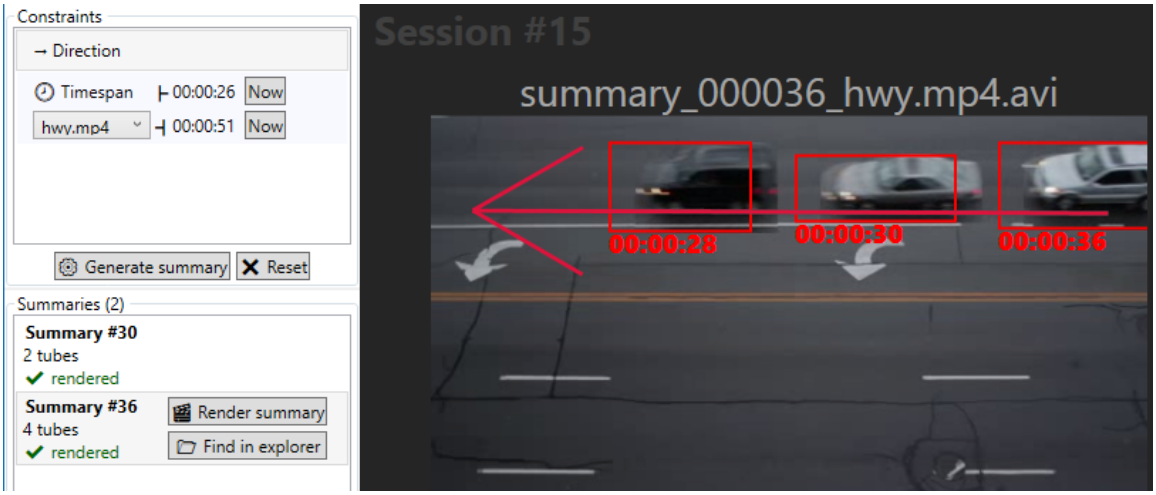

Figure 8.6: A screenshot of a generated summary described in Example 3

8.2.4 Example 4

The aim of this final example is to search for similar vehicles and generate a summary of these vehicle. This example covers the following functionality:

- Similarity based tube retrieval
- Similarity constraint application

Walkthrough

The session from Example 2 is loaded. A van-like vehicle is located (Figure 8.7). A context menu is displayed by a right click and similar tubes are searched for. The results of this search appear in the right panel. When the *Add similarity constraint* menu option is clicked, a constraint corresponding to the results of the search is created. A resulting summary is generated, containing only delivery vehicles similar to the source vehicle, as can be seen in 8.8.

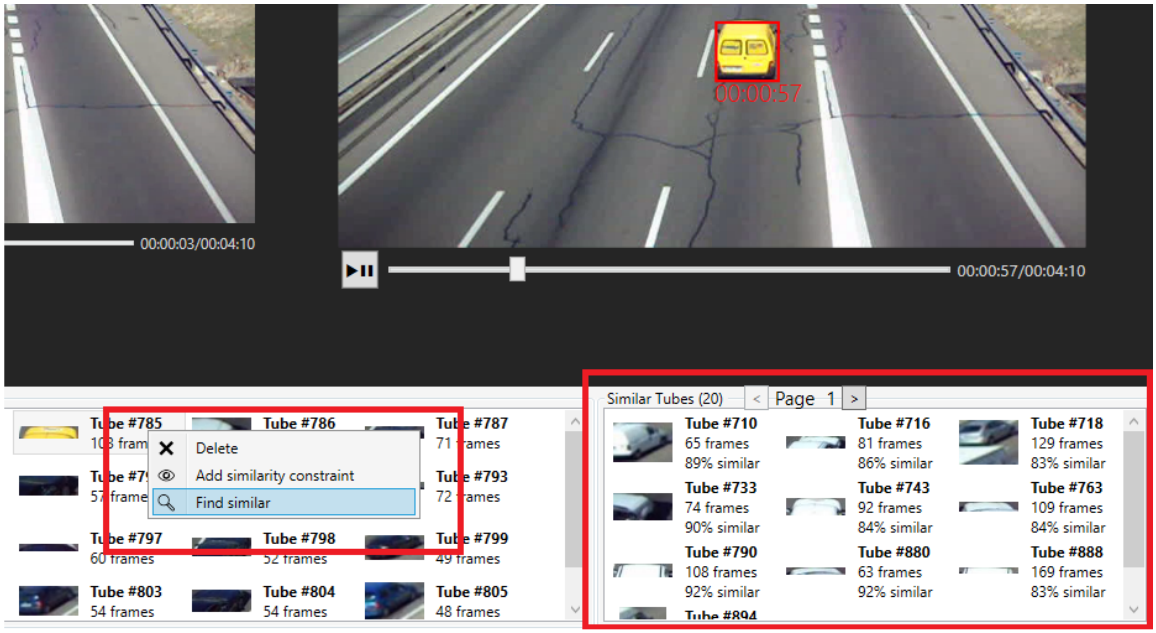

Figure 8.7: A screenshot of context menu and search results described in Example 4

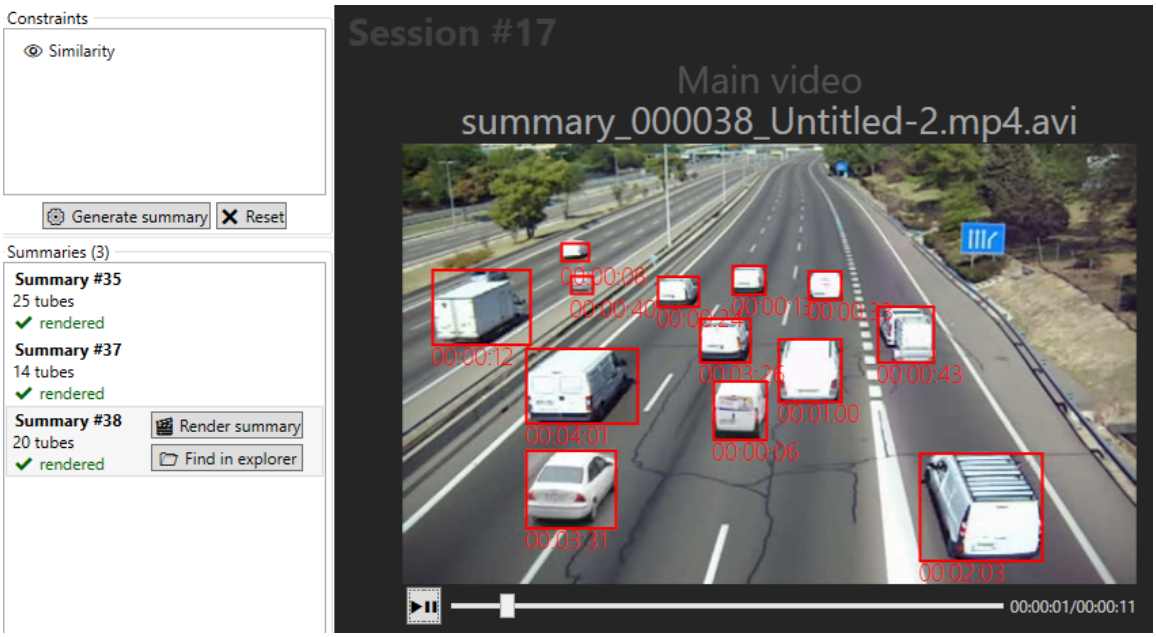

Figure 8.8: A screenshot of resulting summary containing after the application of similarity constraint described in Example 4

8.3 Tests and measurements

This section documents performance of the implemented system. The entire development and testing process was done on a mid-range desktop computer having following specifications:

CPU Intel Skylake Core i5 6600K - 4 core/4 thread processor clocked at 4.5GHz

RAM 16GB of DDR4

GPU NVidia GTX 1060 6GB

No explicit GPU acceleration was implemented.

8.3.1 Testing methodology

Testing consisted of timed execution of individual processes of the summarization process. Following values were measured:

Value	**Meaning**
L_{In}	Length of the input video
N_{Tube}	Number of tubes detected in the video
T_{Ana}	Time spent on input analysis - including BoW histogram generation
T_{Gen}	Time spent on generation of a summary containing all detected tubes
T_{Ren}	Time spent on rendering of the output video
L_{Out}	Length of the output video

Table 8.1: Tracked values

A high activity video (rougly 1.5 new activities per second) was chosen for testing purposes. A 60, 30 and 15 minute variants of this video were created, containing a looped version of the base video. Resolution of the input video was 800x480 and the processing size was set to 540 pixels on the long side, resulting in a resultion of 540x324. Framerate of the video was fixed at 30 frames per second with an average bitrate of 7200kbps.

8.3.2 Tesing results and consequences

Following table presents the experimentally obtained values defined in the previous section.

File	**vert.mp4**	**vert15.mp4**	**vert30.mp4**	**vert60.mp4**
L_{In}	00:04:10	00:15:00	00:30:00	01:00:00
N_{Tube}	438	1532	3051	6109
T_{Ana}	00:02:39	00:08:56	00:18:54	00:36:13
T_{Gen}	00:00:04	00:00:24	00:01:24	00:04:17
T_{Ren}	00:02:59	00:17:12	00:42:48	01:21:08
L_{Out}	00:02:05	00:07:36	00:14:35	00:29:00

Table 8.2: Measured results

Several conclusions can be made about the results. The main goal of summarization was achieved, with the generated videos being considerably shorter than the input videos

with an average of 50% reduction of the input videos' length. The length of the analysis process scaled linearly with the length of the input video, keeping at around 1.6x the speed of source video, which means that the analysis is capable of running in real time. It is important to keep in mind that this also includes the costly tube description process. Generation of the summary's tube arrangement, was almost immediate for numbers of tubes ranging in hundreds, and scaled approximately according to a second order polynomial with increasing number of tubes, evident from Figure 8.9. While the algorithm may not be optimal, the results are satisfactory, providing the results for large sets of tubes (ranging in single thousands) in a matter of minutes.

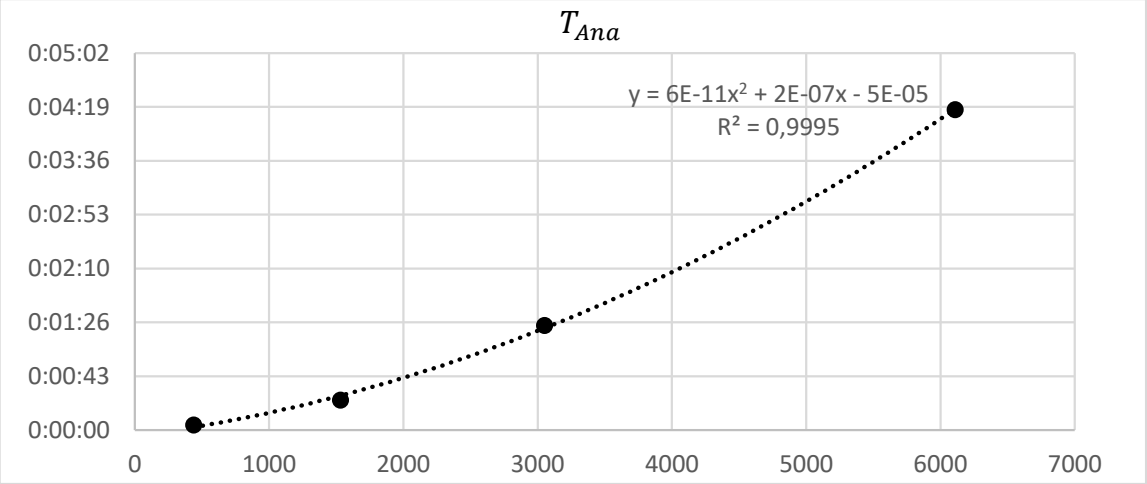

Figure 8.9: Graph of the analysis time and tube count relationship

The actual rendering, however was slower than expected. While bearable for at relatively low number of tubes it was too slow for comfortable usage when processing longer summaries. This is caused mainly by the solution's resource intensive approach, constantly accessing the database as wall as the input source file, each individual tube's image data. The memory usage of this portion was satisfactory, peaking at around 580MB. This part of the system is a great candidate for further optimization.

8.3.3 Discovered implementation problems

A severe problem was discovered during the testing period of the analysis phase. The current version of used OpenCV wrapper, the Emgu CV, contains a memory leak in the implementation of a Disposal routine of all of it's tracker classes. Since the wrapper works by calling the native C/C++ code, the unmanaged memory allocated within this code needs to be managed by the wrapper itself. The wrapper implements the *Dispose* pattern, for this purpose. This method, should by definition of the .NET framework do the following:

> *„Use this method to close or release unmanaged resources such as files, streams, and handles held by an instance of the class that implements this interface. By convention, this method is used for all tasks associated with freeing resources held by an object, or preparing an object for reuse.“* [8]

The wrapper's failure to comply with this contract was verified by a simple experiment. An infinite *while* loop containing two statements was launched. First, an class implementing

the *Tracker* base class was instantialized. Second, the *Dispose* method of this class was called. This led to a gradual increase in the amount of allocated unamanged memory, until the object size limit was reached, resulting in a crash. For control purposes, a different disposable object was used. When placed in this loop, the application was able to run indefinitelly.

This problem will cause a crash during the analysis phase, if a file having the activity frequency comparable to the tested video and the duration of more than about 1,5 hours is analyzed.

8.4 Limitations and known problems

As much as I have tried to create the best possible version of my system, some problems and limitation are known and need to be acknowledged.

Application

- The graphical user interface exists primarily for presentational purposes of the summarization process. The implemented system was not created as a commercial product, neither was it created to fulfill any kind of formal requirements. It is a mere proof of concept of high-level, object-based video summarization. This means that some inconsistencies are bound to be present in the graphical user interface, as no extensive UI testing was done. Each feature was tested to work on its own and in conjunction with related features, during the implementation and preparation of the presentational materials, including this text, but I believe that there is a high possibility of conflicts arising from random combinations of user inputs.
- The VLC library used for playback tends to fail to display the video in some case. This is a known problem described on the developer's page and an application restart is necessary if this occurs.
- The application currently does not support multiple users simultaneously manipulating the data. This was not a requirement and the behavior is not defined. This problem could be solved using the existing session system, making a session and its related tubes and files read-only for other users, until the first user finishes.
- The databse layer is expected to be always accessible by the application and a failure to fulfill this assumption will lead to undefined behaviour.
- Due to the verified memory leak in the implementation of used OpenCV wrapper, the analysis of long, activity-heavy videos will fail, even though the algorithm is sound. This was identified during the testing period and I was not able to produce a workaround in time.

Input analysis

- As mentioned at the start of this chapter, the video analysis process was designed and implemented with traffic videos in mind. This means that different types of videos may produce unsatisfactory results.
- As mentioned in chapter 5, the analysis process assumes that the camera does not move during the entire length of the recording. Should the camera move,

a large number of false movements will be incorrectly tracked, simply because of being, correctly classified as moving objects, since they have changed their positions. In order to solve this problem, additional step of image registration would need to be introduced into the analysis pipeline, increasing its complexity.

- Quality of the Bag of Words histogram generated during analysis process is dependent on the quality of vocabulary used.
- Slow moving objects may become „fused“ with the background, because of the selection of the background subtraction method. This comes back to the assumptions made regarding the type of input videos the system is designed to analyze.
- Input video files are assumed to be without corruptions and to not change their location or names after entering the system. In order to address this, a video file relocation routine could be implemented and added into the GUI, letting the user navigate to the new location of the file, updating the stored metadata.

Summary generation

- The first-fit approach to summary generation produces short videos, but the resulting time arrangement is in not necessarily optimal. A more advanced algorithm could be devised to improve this part of the system.

Output rendering

- Currently, the size of the rendered output video is restricted to the working resolution it was analyzed at. Scaling could be introduced into key parts of the rendering and filtering processes in order to solve this problem. I have decided that this is not an important issue, since the actual content of the resulting summary is the main point of interest.
- The edges of activities exhibit hard seams. This is because the visual fidelity of the output was not a primary concern. This could be partly improved by application of advanced image blending at the edges of rendered slices. Furthermore, the entire foreground masks could be stored during the analysis process in order to improve the blending. This would introduce additional spatial and temporal complexity to the entire process and was omitted from the design of the system for efficiency purposes. Simple rectangles are used because they can be easily stored and compared.

Chapter 9

Conclusion

The primary goal of this project was to produce a system capable of video summarization. In order to achieve this, I had set out to understand the process first.

After the initial research I was able to present the motivation and various possible applications of video summarization. I have familiarized myself with existing works of Prof. Peleg, a pioneer in the field. I have researched the various approaches to video summarization, until I have identified and defined the type of summarization that is suitable for this project.

I have identified and taken a closer look into the inner workings of the building blocks provided by the OpenCV library. I have briefly explained the theoretical background of their key parts necessary for comprehension of this text.

The problem of video summarization was defined, and its various subproblems were identified. This was done in terms of its inputs, outputs and processes of the individual phases of the main problem.

I have proposed a solution in the form of a system, dealing with the individual problems defined previously. The design of this system was described in terms of its individual parts. The concepts of Tubes and related entities were defined. A process of extraction and description of these tubes was devised. A database system for storage of these entities was designed and deployed. Filtering and retrieval processes were designed for stored information. An application was designed to provide the user with means of interaction with stored information and initiation of the summarization process. I have designed the graphical user interface of this application, creating mockups and verbal descriptions of its elements.

Having designed the system, I have chosen the implementation tools and platforms. Many of these were changed during the development in order to increase productivity. The resulting system was comprised of a class library, a database, console application and a GUI application. The system was successfully implemented and was able to successfully produce a video summary from the provided files.

This was exemplified by a set of scenarios walking the reader through the process of creation of different types of summaries. These examples covered the extent of the implemented functionality.

The performance and resource requirements of the system were measured and presented. The overall viability of the system was assessed, and weak points were identified, along with brief ideas for their improvement.

Bibliography

[1] Bar, A.: Smart Thread Pool.
Retrieved from:
https://www.codeproject.com/Articles/7933/Smart-Thread-Pool

[2] Bouwmans, T.: *Recent Advanced Statistical Background Modeling for Foreground Detection: A Systematic Survey*. Laboratoire MIA, Université de La Rochelle. 2011.
Retrieved from: https://www.academia.edu/938519/

[3] Csurka, G.; Dance, C. R.; Fan, L.; et al.: *Visual Categorization with Bags of Keypoints*. Xerox Research Centre Europe. 2004.
Retrieved from:
https://www.cs.cmu.edu/~efros/courses/LBMV07/Papers/csurka-eccv-04.pdf

[4] KaewTraKulPong, P.; Bowden, R.: *An Improved Adaptive Background Mixture Model for Realtime Tracking with Shadow Detection*. Vision and Virtual Reality group, Department of Systems Engineering, Brunel University. 2002.
Retrieved from: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.12.3705&rep=rep1&type=pdf

[5] Kalal, Z.; Mikolajczyk, K.; Matas, J.: *Forward-backward error: Automatic detection of tracking failures*. International Conference on Pattern Recognition. 2010.
Retrieved from: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.231.4285&rep=rep1&type=pdf

[6] Leskovec, J.; Rajaraman, A.; Ullman, J.: *Mining of Massive Datasets*. Cambridge University Press. 2011. ISBN 1107015359 9781107015357.

[7] Lowe, D. G.: *Distinctive Image Features from Scale-Invariant Keypoints*. Computer Science Department University of British Columbia. 2004.
Retrieved from:
https://people.eecs.berkeley.edu/~malik/cs294/lowe-ijcv04.pdf

[8] Microsoft: IDisposable.Dispose Method documentation.
Retrieved from:
https://msdn.microsoft.com/en-us/library/system.idisposable.dispose(v=vs.110).aspx

[9] Moersch, J.: QuickConverter.
Retrieved from: https://github.com/JohannesMoersch/QuickConverter

[10] Newton, E.: Command Line Parser Library for CLR and NetStandard.
Retrieved from: https://github.com/commandlineparser/commandline

[11] Perone, C. S.: Machine Learning :: Cosine Similarity for Vector Space Models (Part III).
Retrieved from: http://blog.christianperone.com/2013/09/machine-learning-cosine-similarity-for-vector-space-models-part-iii/

[12] Pritch, Y.; Ratovitch, S.; Hendel, A.; et al.: *Clustered Synopsis of Surveillance Video*. School of Computer Science and Engineering The Hebrew University of Jerusalem. 2009.
Retrieved from:
http://www.vision.huji.ac.il/video-synopsis/avss09-ClusteredSynopsis.pdf

[13] Pritch, Y.; Rav-Acha, A.; Gutman, A.; et al.: *Webcam Synopsis: Peeking Around the World*. School of Computer Science and Engineering The Hebrew University of Jerusalem. 2007.
Retrieved from:
http://www.vision.huji.ac.il/video-synopsis/iccv07-webcam.pdf

[14] Pritch, Y.; Rav-Acha, A.; Peleg, S.: *Nonchronological Video Synopsis and Indexing*. School of Computer Science and Engineering The Hebrew University of Jerusalem. 2008.
Retrieved from:
http://www.vision.huji.ac.il/video-synopsis/pami08-synopsis.pdf

[15] Rav-Acha, A.; Pritch, Y.; Peleg, S.: *Making a Long Video Short: Dynamic Video Synopsis*. School of Computer Science and Engineering The Hebrew University of Jerusalem. 2006.
Retrieved from:
http://www.vision.huji.ac.il/video-synopsis/cvpr06-synopsis.pdf

[16] Sinha, U.: SIFT: Theory and Practice.
Retrieved from: http://aishack.in/tutorials/sift-scale-invariant-feature-transform-introduction/

[17] Trevino, A.: Introduction to K-means Clustering.
Retrieved from: https://www.datascience.com/blog/k-means-clustering

[18] About us - BriefCam.
Retrieved from: http://briefcam.com/about-us/#history

[19] OpenCV library.
Retrieved from: https://opencv.org/

[20] OpenCV: TrackerMedianFlow Class Reference.
Retrieved from:
https://docs.opencv.org/3.4.0/d7/d86/classcv_1_1TrackerMedianFlow.html

Appendix A

Contents of DVD

The attached DVD contains the following items:

- Digital version of this text.
- Digital version of this text intended for printing.
- A Visual Studio Solution containg the source code of the implemented software.
- A Readme file describing the steps of launching the software.
- Reference inputs.
- Reference outputs.
- A video demonstration of the implemented features.
- A digital version of the poster presenting achieved results.